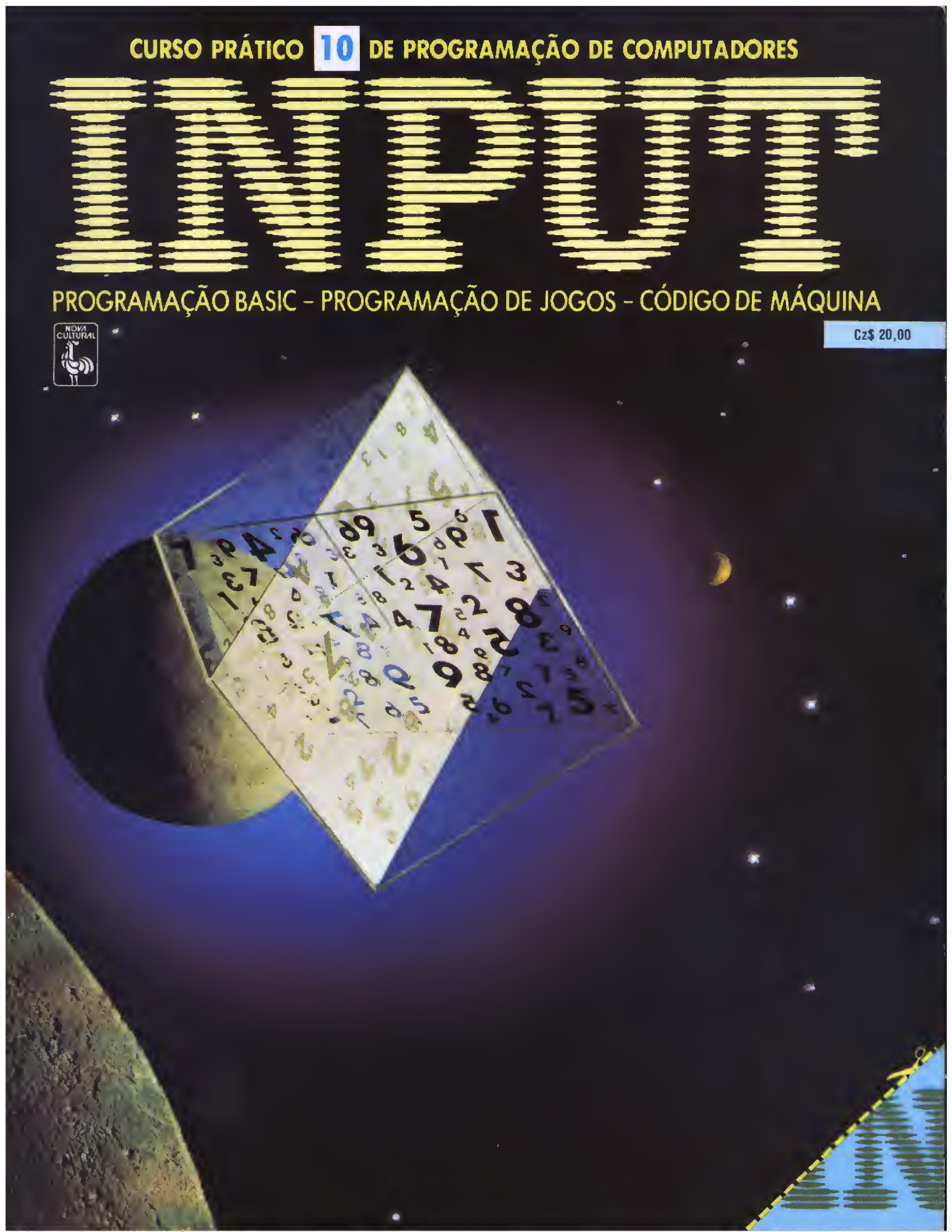
CURSO PRÁTICO 10 DE PROGRAMAÇÃO DE COMPUTADORES

INPUT

PROGRAMAÇÃO BASIC - PROGRAMAÇÃO DE JOGOS - CÓDIGO DE MÁQUINA

NOVA CULTURAL

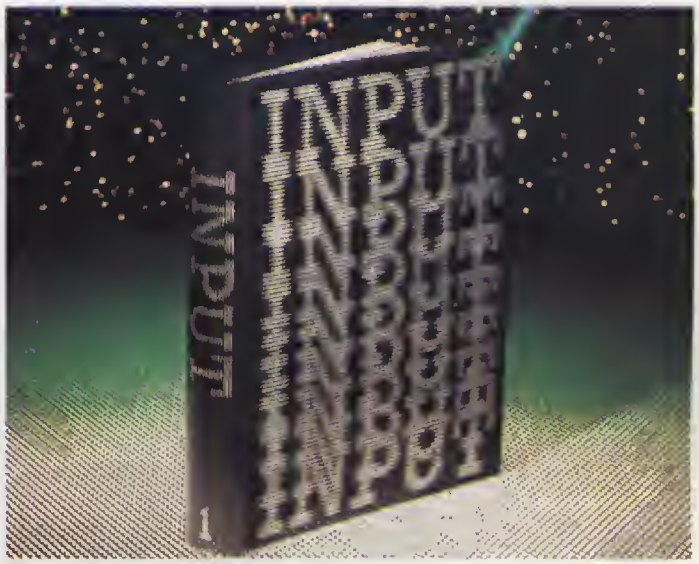
Cz$ 20,00



INPUT

Vol. 1 N.º 10

NESTE NÚMERO



PLANO DA OBRA

"INPUT" é uma obra editada em fascículos semanais, e cada conjunto de 15 fascículos compõe um volume. A capa para encadernação de cada volume estará à venda oportunamente.

COMPLETE SUA COLEÇÃO

Exemplares atrasados, até seis meses após o encerramento da coleção, poderão ser comprados, a preços atualizados, da seguinte forma: **1. Pessoalmente** — por meio de seu jornaleiro ou dirigindo-se ao distribuidor local, cujo endereço poderá ser facilmente conseguido junto a qualquer jornaleiro de sua cidade. Em São Paulo os endereços são: Rua Brigadeiro Tobias, 773 (Centro); Av. Industrial, 117 (Santo André); e, no Rio de Janeiro: Rua da Passagem, 93 (Botafogo). **2. Por carta** — Poderão ser solicitados exemplares atrasados também por carta, que deve ser enviada para DINAP — Distribuidor Nacional de Publicações — Números Atrasados — Estrada Velha de Osasco, 132 (Jardim Tereza) — CEP 06000 — Osasco — São Paulo. **3. Por telex** — Utilize o nº (011) 33670 ABSA. Em **Portugal**, os pedidos devem ser feitos à Distribuidora Jardim de Publicações Ltd. — Qta. Pau Varais, Azinhaga de Fetais — 2685, Camarate — Lisboa; Tel. 257-2542 — Apartado 57 — Telex 43 069 JARLIS P.
Não envie pagamento antecipado. O atendimento será feito pelo reembolso postal e o pagamento, incluindo as despesas postais, deverá ser efetuado ao se retirar a encomenda na Agência do Correio. **Atenção**: Após seis meses do encerramento da coleção, os pedidos serão atendidos, dependendo da disponibilidade de estoque. **Obs.**: Quando pedir livros, mencione sempre o título e/ou o autor da obra, além do número da edição.

COLABORE CONOSCO

Encaminhe seus comentários, críticas, sugestões ou reclamações ao Serviço de Atendimento ao Leitor — Caixa Postal 9442, São Paulo — SP.

Editor
VICTOR CIVITA

REDAÇÃO

Diretora Editorial: Iara Rodrigues

Editor chefe: Paulo de Almeida
Editor de texto: Cláudio A.V. Cavalcanti
Editor de Arte: Eduardo Barreto
Chefe de Arte: Carlos Luiz Batista
Assistentes de Arte: Ailton Oliveira Lopes, Dilvacy M. Santos, José Maria de Oliveira, Grace A. Arruda, Monica Lenardon Corradi
Secretária de Redação/Coordenadora: Stefania Crema
Secretários de Redação: Beatriz Hagström, José Benedito de Oliveira Damião, Maria de Lourdes Carvalho, Marisa Soares de Andrade, Mauro de Queiroz
Secretário Gráfico: Antonio José Filho

COLABORADORES
Consultor Editorial Responsável: Dr. Renato M.E. Sabbatini (Diretor do Núcleo de Informática Biomédica da Universidade Estadual de Campinas)
Execução Editorial: DATAQUEST Assessoria em Informática Ltda. Campinas, SP.

Tradução: Maria Fernanda Sabbatini

Adaptação, programação e redação: Abílio Pedro Neto, Alufsio J. Dornellas de Barros, Marcelo R. Pires Therezo, Raul Neder Porrelli,
Coordenação geral: Rejane Felizatti Sabbatini
Assistente de Arte: Dagmar Bastos Sampaio
COMERCIAL
Diretor Comercial: Roberto Martins Silveira
Gerente Comercial: Flávio Ferrucio Maculan
Gerente de Circulação: Denise Maria Mozol

PRODUÇÃO
Gerente de Produção: João Stungis
Coordenador de Impressão: Atílio Roberto Bonon
Preparador de Texto/Coordenador: Eliel Silveira Cunha

Preparadores de Texto: Ana Maria Dilguerian, Antonio Francelino de Oliveira, Karina Ap. V Grechi, Levon Yacubian, Maria Teresa Galluzzi, Pauto Felipe Mendrone
Revisor/Coordenador: José Maria de Assis
Revisoras: Conceição Aparecida Gabriel, Isabel Leite de Camargo, Ligia Aparecida Ricetto, Maria do Carmo Leme. Monteiro, Maria Luiza Simões, Maria Teresa Martins Lopes.

Esta obra foi composta na AM Produções Gráficas Ltda. e impressa na Divisão Gráfica da Editora Abril S.A.

REÚNA SEUS DADOS EM GRÁFICOS

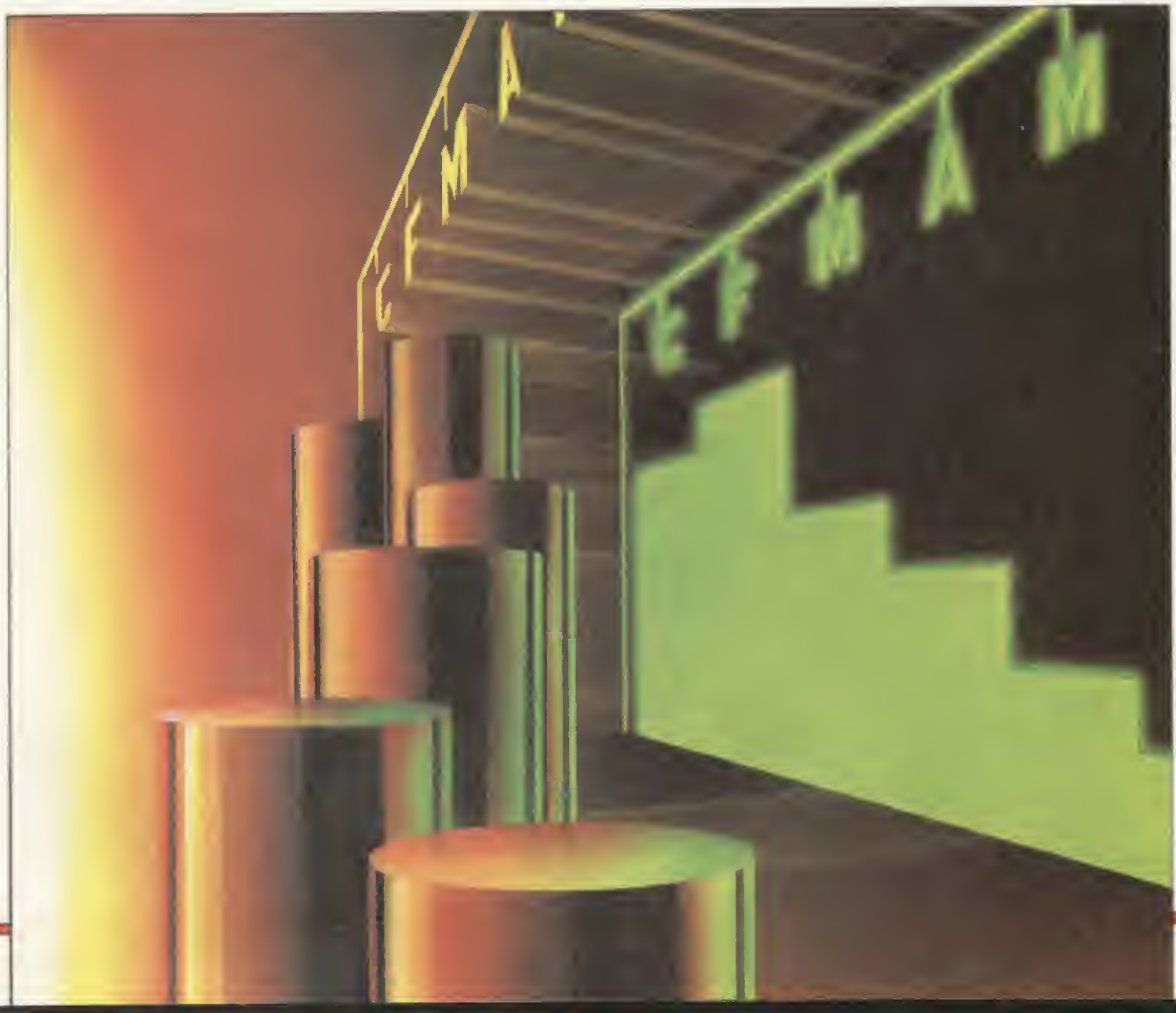
- COMO FUNCIONA O PROGRAMA
- ENTRADA DAS INFORMAÇÕES
- CORREÇÕES
- ELABORAÇÃO DOS GRÁFICOS
- ARMAZENAGEM DOS DADOS

Digite os números e deixe o micro convertê-los em histogramas coloridos e bem-feitos. Por meio deles, será fácil comparar valores diversos ou verificar suas tendências temporais.

Você já deve ter visto alguns anúncios de programas para aplicações comerciais, nos quais se destaca a elaboração de gráficos sobre estatísticas de vendas, evolução de cotação das ações na Bolsa, etc. Esse tipo de informação, que em geral envolve uma quantidade enorme de números, torna-se de fácil compreensão quando representado por diagramas ou gráficos. E se preparar um gráfico manualmente é trabalhoso, para um computador comercial trata-se de tarefa simples e rápida.

A capacidade de representar informações não é, entretanto, atributo exclusivo dos computadores comerciais. Ela estende-se, também, aos computadores domésticos, constituindo-se em uma de suas áreas de aplicação prática. Todos os computadores apresentados aqui possuem capacidade numérica e gráfica que os habilita a efetuar esse tipo de trabalho, com graus variáveis de precisão e de detalhamento gráfico.

Dificilmente o usuário médio de computadores domésticos precisará lidar com a vasta quantidade de informações geradas, por exemplo, por uma pequena empresa. Mas numerosas questões que lhe interessam diretamente podem ser analisadas de maneira proveitosa em um microcomputador. Por exemplo, a relação entre seus rendimentos e gastos



ao longo de um ano; o peso específico, em períodos distintos, dos diversos itens que compõem seu orçamento; as variações mensais de sua capacidade de poupança.

Além dessas áreas, cujas aplicações se assemelham às comerciais, existem várias outras, sobretudo relacionadas com algum tipo de passatempo, que podem ter seus dados analisados e representados graficamente: a freqüência ao clube local ou a espetáculos teatrais em determinado período, a evolução do número de itens em coleções de livros, discos ou selos e estatísticas de resultados esportivos.

O programa apresentado serve para preparar, rapidamente, um gráfico de barras — ou histograma —, mostrando as variações temporais de algum tipo de valor. Como os eixos do gráfico (horizontal e vertical) ajustam-se automaticamente, pode-se trabalhar com dados semanais, mensais, anuais ou em qualquer outra unidade de tempo.

O intervalo máximo de valores utilizado pelo programa depende do seu computador. Para o TRS-Color, esses valores situam-se entre +99 e −99. Para o Spectrum, entre +1000 e −1000, mais ou menos. Já para o Apple e o MSX, tais valores podem ter qualquer amplitude — unidades, dezenas, centenas, milhares e até milhões, se sua aplicação assim requerer.

ENTRADA DOS DADOS

Ao executar o programa, este apresentará na tela uma lista das opções disponíveis (o menu). Se for selecionada a opção para a entrada de dados, o programa pedirá os títulos dos dois eixos, apresentando-os no momento em que traçar o gráfico.

Ao entrar com os títulos dos eixos, tenha sempre o cuidado de digitá-los corretamente e na ordem pedida. O eixo horizontal (X) representa o tempo: semanas, meses, anos ou qualquer outra unidade escolhida. O eixo vertical (Y) representa os valores dados às barras — cruzados, temperatura, número de discos, etc. A altura de cada uma das barras será proporcional a este valor.

Em seguida, o programa pede o número de barras que deve apresentar. O número máximo possível depende, basicamente, da capacidade gráfica do seu computador. O Spectrum, por exemplo, apresenta no máximo 25 barras; no TRS-Color, este número passa a 26; o Apple e o MSX têm capacidade para apresentar até 200 barras, mas seu limite prático é de cerca de 65. Acima deste valor, as bárras ficam muito finas ou irregulares, constituindo um quadro visualmente pouco claro.

As perguntas seguintes feitas pelo programa referem-se aos valores dos dados. São apresentados na tela os números seqüenciais das barras e uma pergunta sobre o valor a ser atribuído a cada uma. Esse valor, positivo ou negativo, é, novamente, limitado pela capacidade numérica do seu computador. No TRS-Color, os valores vão de −999 a +999 e, no Spectrum, de −1000 a +1000. O Apple e o MXS, por sua vez, trabalham com valores de qualquer magnitude, já que os gráficos são montados com o uso de escalas.

Se seu computador é um Spectrum ou um TRS-Color, e você deseja representar valores que estão fora da capacidade do programa, a solução é fornecer os dados divididos ou multiplicados por algum fator de conversão (por exemplo, em unidades de centenas, milhares, milhões, etc.).

Quando o último valor for introduzido, surgirá novamente a lista inicial de opções. Você poderá, então, escolher entre modificar algum valor digitado ou partir para a apresentação do gráfico. Se optar pela modificação, os valores que digitou serão apresentados na tela. Siga as instruções para corrigir o valor desejado.

Quando estiver satisfeito com os valores, selecione a opção que permite a apresentação do gráfico. Os usuários do Apple e do MSX possuem apenas uma maneira de apresentar seus gráficos. Usuários do TRS-Color e do Sinclair Spectrum podem escolher entre um gráfico em escala e um gráfico em representação real.

A opção de um gráfico escalonado apresenta o diagrama de barras com as dimensões, ao longo do eixo y, arredondadas para um valor máximo de 10, 100, 1000, etc., dependendo do valor máximo dos dados. Conforme este valor, porém, o gráfico pode não aparecer inteiramente na tela.

A opção de um gráfico em representação real mostra o diagrama de barras ocupando a tela inteira, apresentando os valores reais dos dados (não os valores em escala) ao longo do eixo y.

Para maior clareza, as barras apresentadas pelo Apple, pelo MSX e pelo Spectrum são separadas umas das outras por um pequeno espaço ou uma barra colorida. O TRS-Color apresenta as barras coloridas alternadamente de cinza e amarelo, para os valores positivos, e de vermelho e laranja, para os valores negativos.

COMO FUNCIONA O PROGRAMA

O programa de elaboração de histogramas tem três rotinas principais:

- entrada dos dados que serão exibidos no gráfico;
- edição (modificação) dos dados já entrados;
- elaboração do gráfico.

Os dados são armazenados no conjunto A pela primeira rotina. A capacidade desse conjunto é especificada dinamicamente pelo comando **DIM**, após o usuário indicar o número de dados que deseja utilizar. Quando o programa entra na rotina de elaboração do gráfico, o conjunto de dados é examinado, para que se identifiquem os valores mínimo e máximo nele presentes. Esses valores são armazenados nas variáveis **LO** e **HI**, respectivamente. O valor mínimo predefinido é sempre zero; se os dados apresentarem um valor mínimo maior, este será ignorado pelo programa. O valor máximo é arredondado: 10, 100, 1000, 10 000, etc.

Havendo dados negativos no conjunto A (valor mínimo menor que zero), o programa definirá que o gráfico é de tipo 1; senão, de tipo 2. Nos gráficos de tipo 1, a linha de base (eixo dos x) ficará acima do fundo da tela, e as barras correspondentes aos valores negativos serão dirigidas para baixo.

O escalamento dos valores de um eixo é feito segundo uma regra simples, que leva em conta os valores mínimo e máximo, o número de pixels (pontos gráficos) que cabem no eixo e o arredondamento ou não dos rótulos ao longo do eixo. A fórmula de escalamento é:

$$x' = \frac{x - xmin}{xmax - xmin} \cdot pix$$

onde:

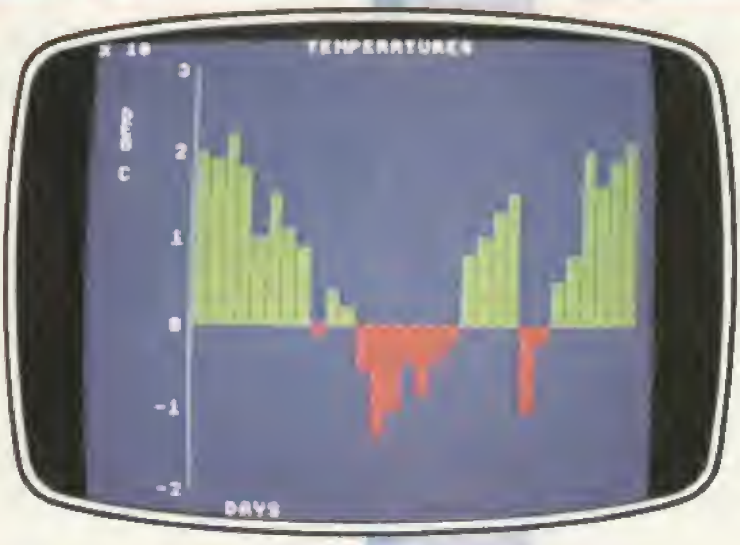
x' = valor escalado
x = valor original
xmin = valor mínimo na escala
xmax = valor máximo na escala
pix = número máximo de pixels no eixo

Em seguida, o gráfico é elaborado. Cada computador utiliza um método distinto para traçar as barras. Se você conhecer um pouco de programação BASIC, poderá, nesse ponto, alterar o programa para fazer coisas diferentes.

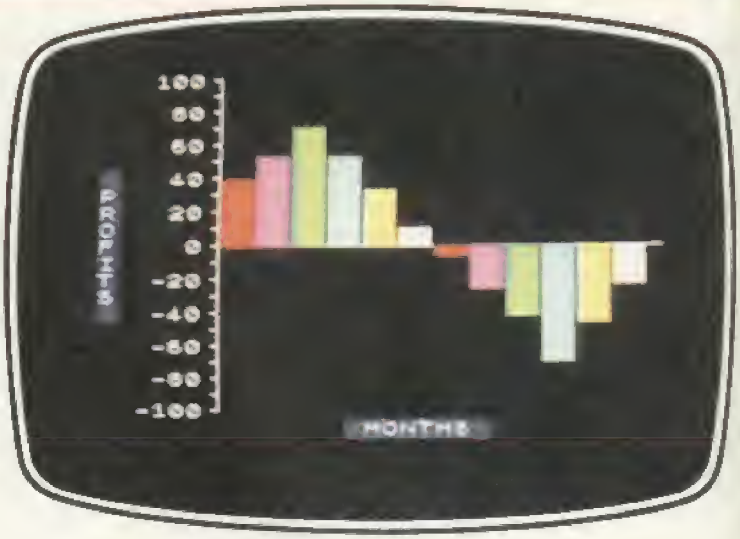
Por exemplo, a versão para o Sinclair Spectrum utiliza o comando **PLOT** para posicionar o cursor gráfico no ponto onde a barra vertical será traçada, e o comando **DRAW**, para traçá-la de uma vez. Se você substituir o comando **DRAW** por um outro **PLOT**, obterá apenas um ponto colorido, correspondente ao topo da barra.

Outra modificação se refere à definição das cores. No Sinclair Spectrum, por exemplo, o comando **INK** determina a cor a ser usada na barra. Note que a variável que se segue a este comando é modificada automaticamente pelo programa, de modo a alterar a cor da barra seguinte. Por meio de um comando **INPUT**, você pode mudar o programa, determinando que a cor seja solicitada ao usuário.

Interessante também é a possibilidade de modificar o programa para armazenar os dados entrados, em fita cassete ou disquete. Nesse caso, o menu inicial deverá ter duas opções adicionais:



MSX: gráfico da variação de temperatura ao longo do ano.

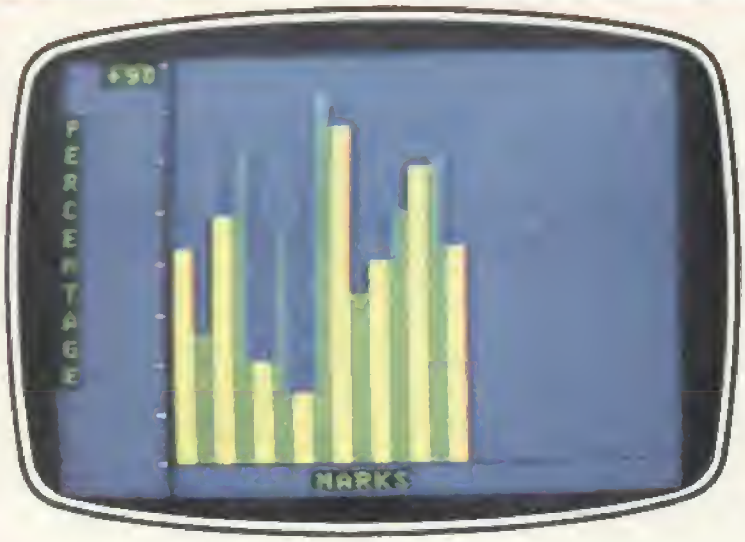


ZX Spectrum: gráfico de lucros e perdas por mês

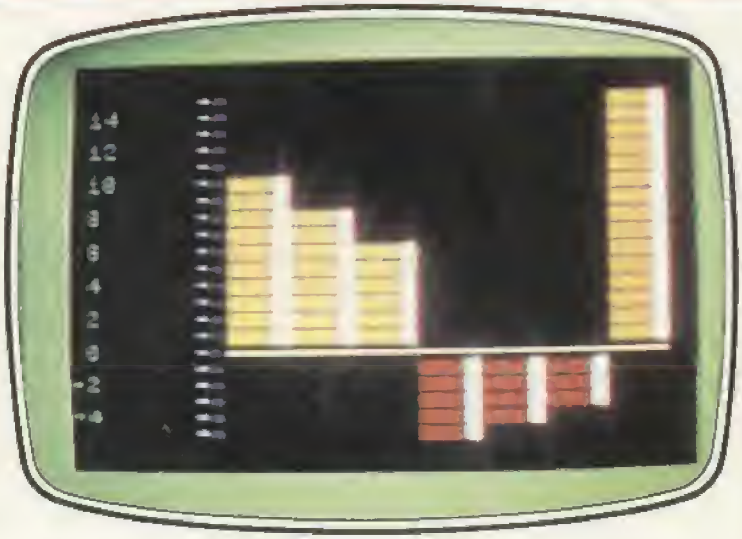
- armazenar dados
- gravar dados

As rotinas correspondentes a estas chamadas deverão ser programadas e adicionadas ao programa. Assim, você poderá manter séries de dados armazenados (por exemplo, os rendimentos da caderneta de poupança ou os gastos mensais com alimentação) e atualizá-los com novos valores sempre que quiser compará-los ou verificar sua tendência por meio do gráfico. Com esta modificação, você não precisará, portanto, digitar os dados anteriores toda vez que executar um programa para elaboração de gráfico.

```
10 PMODE 4,1
20 CLS
30 PRINT @45,"MENU":PRINT @102,
"1- INTRODUZIR DADOS":PRINT @16
6,"2- GRAFICO DE BARRAS":PRINT
@230,"3- VER/CORRIGIR DADOS":PR
INT @294,"4- SAIR DO PROGRAMA"
40 A$=INKEY$:IF A$<"1" OR A$>"4
" THEN 40
50 IF A$="1" AND DA=1 THEN 80
60 ON VAL(A$) GOSUB 1000,2000,3
000,4000
70 GOTO 20
80 PRINT @484,"VOCE TEM CERTEZA
? ";
90 A$=INKEY$:IF A$<>"Y" AND A$<
>"N" THEN 90
100 IF A$="Y" THEN CLEAR 200:A$
="1":GOTO 60
110 GOTO 20
1000 DA=1:CLS:INPUT"NOME DO EIX
O X";X$
1010 X$=LEFT$(X$,32):IF X$="" T
HEN X$="EIXO X"
1020 MD=INT(16-LEN(X$)/2)
1030 INPUT"NOME DO EIXO Y";Y$
1040 Y$=LEFT$(Y$,12):IF Y$="" T
HEN Y$="EIXO Y"
1050 HT=INT(6-LEN(Y$)/2)
1060 INPUT"NUMERO DE BARRAS";NB
1070 NB=INT(NB):IF NB<1 THEN 10
60
1080 BL=INT(26/NB):IF BL<1 THEN
 BL=1
1090 DIM A(NB)
1100 PRINT
1110 FOR K=1 TO NB
1120 PRINT"VALOR DA BARRA";K;:I
NPUT A(K)
1130 NEXT
1140 RETURN
2000 IF DA=0 THEN PRINT @455,"D
ADOS NAO INTRODUZIDOS":FOR K=1
TO 2000:NEXT:RETURN
2010 TP=0:BT=0
2020 FOR K=1 TO NB
2030 IF A(K)>TP THEN TP=A(K)
2040 IF A(K)<BT THEN BT=A(K)
2050 NEXT
2060 IF TP=0 AND BT=0 THEN PRIN
T "TODOS OS VALORES SAO ZERO ":
FOR K=1 TO 2000:NEXT:RETURN
2070 CLS:PRINT @64,"VOCE QUER U
M GRAFICO EM ESCALA (S/N)?"
2080 A$=INKEY$:IF A$<>"S" AND A
$<>"N" THEN 2080
2090 IF A$="N" THEN 2120
2100 IF TP>0 THEN E=INT(LOG(TP)
/LOG(10)):TP=INT(1+TP/(10^E))*1
0^E
2110 IF BT<0 THEN E=INT(LOG(-BT
)/LOG(10)):BT=-INT(1-BT/(10^E))
*10^E
2120 IN=178/(TP-BT)
2130 ST=INT(IN*TP)
2140 T$="":IF TP=0 THEN 2160
2150 IF TP>ABS(BT) THEN E=2*INT
(LOG(TP)/LOG(1000)) ELSE E=2*IN
T(LOG(ABS(BT))/LOG(1000))
2160 T$="+"+MID$(STR$(INT(.5+TP
/10^E)),2):B$="":IF BT=0 THEN 2
180
2170 BS=STR$(INT(.5+BT/10^E))
2180 SL=1:LP=NB:IF NB>26 THEN L
P=26
2190 POKE 179,243:PCLS:POKE 654
75,0:POKE 65477,0:POKE 65479,0
2200 POKE 179,2
2210 LINE(40,0)-(47,191),PRESET
,BF
2220 IF T$="" THEN 2260
2230 FOR K=1 TO LEN(T$):P=ASC(M
ID$(T$,K,1)) AND 63:FOR M=0 TO
11
2240 POKE 1540-LEN(T$)+32*M+K,P
2250 NEXT M,K
2260 IF B$="" THEN 2300
2270 FOR K=1 TO LEN(B$):P=ASC(M
ID$(B$,K,1))AND 63:FOR M=0 TO 1
1
2280 POKE 6916-LEN(B$)+M*32+K,P
2290 NEXT M,K
2300 FOR K=1 TO LEN(X$)
2310 P=ASC(MID$(X$,K,1)) AND 63
2320 FOR M=182 TO 190:POKE 1503
+MD+32*M+K,P:NEXT
2330 NEXT
2340 FOR K=1 TO LEN(Y$)
2350 FOR M=0 TO 11
2360 P=ASC(MID$(Y$,K,1)) AND 63
2370 POKE 1568+384*(K+HT)+32*M,
P
2380 NEXT M,K
2390 FOR K=5 TO 31
2400 POKE 1536+32*ST+K,128
2410 NEXT
2420 FOR K=ST TO 0 STEP -22
2430 POKE 1541+K*32,202
2440 NEXT
2450 FOR K=ST TO 178 STEP 22
2460 POKE 1541+K*32,202
2470 NEXT
2480 FOR K=SL TO LP:CO=(CO+1) A
ND 1
2490 POKE 178,35+CO*64:POKE 179
,3+CO*192
2500 IF INT(A(K)*IN)>0 THEN LIN
E(48+8*(K-SL)*BL,ST-1)-(47+8*(K
```



TRS-Color: gráfico da distribuição das notas em uma classe.



Apple: gráfico dos saldos mensais da poupança.

```
-SL+1)*BL,ST-INT(A(K)*IN)),PSET
,BF
2510 IF FIX(A(K)*IN)<0 THEN LIN
E(48+8*(K-SL)*BL,ST)-(47+8*(K-S
L+1)*BL,ST-FIX(A(K)*IN)),PRESET
,BF
2520 NEXT K
2530 IF LP=NB THEN 2570
2540 SL=LP+1:LP=LP+26:IF LP>NB
THEN LP=NB
2550 IF INKEY$="" THEN 2550
2560 POKE 179,243:LINE(48,0)-(2
55,178),PRESET,BF:GOTO 2390
2570 IF INKEY$="" THEN 2570
2580 RETURN
3000 IF DA=0 THEN PRINT @455,"D
ADOS NAO INTRODUZIDOS":FOR K=1
TO 2000:NEXT:RETURN
3010 SL=1:LP=NB:IF NB>14 THEN L
P=14
3020 CLS:PRINT"BARRA","VALOR"
3030 FOR K=SL TO LP
3040 PRINT K,A(K)
3050 NEXT
3060 IF NB=LP THEN 3130
3070 PRINT @480,"e PARA EDITAR,
 QUALQUER OUTRA   PARA CONTINUA
R";
3080 A$=INKEY$:IF A$="" THEN 30
80
3090 IF A$="E" THEN 3170
3100 SL=LP+1:LP=LP+14
3110 IF LP>NB THEN LP=NB
3120 GOTO 3020
3130 PRINT @480,"e PARA EDITAR,
 QUALQUER OUTRA   PARA RETORNAR
";
3140 A$=INKEY$:IF A$="" THEN 31
40
3150 IF A$="E" THEN 3170
3160 RETURN
3170 CLS:PRINT:INPUT"NUMERO DA
ENTRADA A SER EDITADA ?";E
3180 E=INT(E):IF E<1 OR E>NB TH
EN 3170
3190 PRINT:PRINT"NOVO VALOR DA
ENTRADA";E;:INPUT A(E)
3200 GOTO 3020
4000 CLS:PRINT @37,"VOCE TEM CE
RTEZA (S/N)?"
4010 A$=INKEY$:IF A$<>"S" AND A
$<>"N" THEN 4010
4020 IF A$="N" THEN RETURN
```

```
 10 LET g=0: POKE 23609,20:
 POKE 23658,8
100 BORDER 7: PAPER 7: INK 0:
CLS
110 PRINT  BRIGHT 1; PAPER 3;
INK 7;AT 4,10;" O P C O E S "
120 PRINT  BRIGHT 1;AT 7,4;" 1
- INTRODUZIR NOVOS DADOS "
130 PRINT  BRIGHT 1;AT 9,4;" 2
- VER / EDITAR DADOS      "
140 PRINT  BRIGHT 1;AT 11,4;"
3- GRAFICO EM ESCALA      "
145 PRINT  BRIGHT 1;AT 13,4;"
4- GRAFICO EM TELA CHEIA  "
150 PRINT  BRIGHT 1; FLASH 1;
INK 2;AT 16,9;" SELECIONE OPCA
O "
160 IF INKEY$="" THEN  GOTO
160
170 LET A$=INKEY$: IF A$<"1"
OR A$>"4" THEN  GOTO 160
180 GOSUB VAL A$*1000: GOTO
100
500 REM **ROTINA PARA ENTRADA
DE DADOS NUMERICOS**
510 INPUT (w$); LINE a$: IF
LEN a$=0 THEN  GOTO 510
520 FOR j=1 TO LEN a$
540 IF (a$(j)>="0" AND a$(j)<=
"9") OR a$(j)="." OR a$(j)="-"
THEN  NEXT j: LET v=VAL a$:
RETURN
 550 GOTO 510
1000 REM **ROTINA PARA ENTRADA
DE DADOS**
1010 BORDER 1: PAPER 1: INK 7:
CLS
1020 INPUT "NOME DO EIXO X? ";
LINE x$
1030 PRINT  INVERSE 1;AT 0,0;"
";x$;" "
1040 INPUT "NOME DO EIXO Y? ";
LINE y$
1050 PRINT  INVERSE 1;AT 0,16;"
";y$;" "
1060 LET w$="QUANTO "+x$+" (1 A
 25)? ": GOSUB 500
1070 IF v<1 OR v>25 OR v<>INT v
 THEN  GOTO 1060
1090 LET z=v: DIM a(z)
1100 FOR k=1 TO z
1110 LET w$="DIGITE DADOS PARA
"+STR$ k+" ": GOSUB 500
1120 LET a(k)=v
1130 PRINT k,a(k)
1140 NEXT k: LET g=1: PAUSE 50:
 RETURN
2000 REM **ROTINA PARA EDICAO D
E DADOS**
2010 BORDER 2: PAPER 2: INK 7
2020 LET cn=1
2025 CLS : PRINT  PAPER 6; INK
2;AT 0,0;x$,y$;TAB 31;" "
2030 PRINT cn,a(cn)
2035 PRINT #1; PAPER 6; INK 2;A
T 0,0;"EDIT para alterar valor
corrente Qualquer tecla para co
ntinuar   "
2040 PAUSE 0
2050 IF INKEY$="" THEN  GOTO 20
50
2060 LET c$=INKEY$
2070 IF c$=CHR$ 7 THEN  GOSUB 2
500
2080 IF cn=z THEN  PRINT  PAPER
 6; INK 2;"FIM DOS DADOS": PAUS
E 100: RETURN
2090 LET cn=cn+1: IF cn=21 THEN
  GOTO 2025
2100 GOTO 2030
2500 LET w$="INTRODUZIR NOVO VA
LOR PARA "+STR$ cn+" ": GOSUB 5
00
2510 LET a(cn)=v: PRINT  PAPER
6; INK 2;cn,a(cn);TAB 31;" ": R
ETURN
```

```
3000 REM **GRAFICO EM ESCALA**
3010 BORDER 0: PAPER 0: INK 7:
CLS : LET hi=0: LET lo=0
3020 FOR k=1 TO z
3030 IF a(k)>hi THEN  LET hi=a(
k)
3040 IF a(k)<lo THEN  LET lo=a(
k)
3050 NEXT k
3060 LET type=2: LET org=4
3070 IF lo<0 THEN  LET type=1:
LET org=84
3080 LET h=hi: IF ABS lo>hi THE
N  LET h=ABS lo
3090 LET ra=hi-lo
3100 IF h<=1 THEN  LET hi=1: GO
TO 3150
3110 IF h<=10 THEN  LET hi=10:
GOTO 3150
3120 IF h<=100 THEN  LET hi=100
: GOTO 3150
3130 IF h<=1000 THEN  LET hi=10
00: GOTO 3150
3140 IF h<=10000 THEN  LET hi=1
0000
3150 LET wd=INT (25/z)
3160 IF type=1 THEN  LET ft=80/
hi
3170 IF type=2 THEN  LET ft=162
/hi
3180 PLOT 56,org: DRAW 198,0
3190 PLOT 55,4: DRAW 0,160
3200 FOR n=4 TO 168 STEP 8: PLO
T 52,n: DRAW 3,0: NEXT n
3220 PRINT #1; PAPER 1;AT 0,14;
" ";x$;" "
3225 LET z$=" "+y$+" "
3230 FOR n=1 TO LEN z$
3240 PRINT AT n+(19-LEN y$)/2,0
; PAPER 1;z$(n)
3250 NEXT n
3255 LET dc=1
3260 IF type=1 THEN  GOTO 3320
3270 FOR n=hi TO 0 STEP -(hi/10
)
3275 IF n<.01 THEN  LET n=0
3280 LET n$=STR$ n
3290 PRINT AT dc,(6-LEN n$);n
3295 LET dc=dc+2
3300 NEXT n
3310 GOTO 3400
3320 FOR n=hi TO -hi STEP -(hi/
5)
3330 IF n<.01 AND n>-.01 THEN
LET n=0
3340 LET n$=STR$ n
3350 PRINT AT dc,(6-LEN n$);n
3360 LET dc=dc+2
3370 NEXT n
3400 LET ink=1
3410 FOR n=1 TO z
3420 LET cm=org
3430 LET ink=ink+1: IF ink=8 TH
EN  LET ink=2
3440 INK ink
3450 FOR m=1 TO (a(n)*ft) STEP
SGN a(n)
3460 PLOT 56+(n-1)*wd*8,cm: DRA
W wd*8-2,0
3470 LET cm=cm+SGN a(n)
3480 NEXT m
3490 NEXT n
3500 IF INKEY$<>"" THEN  GOTO 3
500
3510 IF INKEY$="" THEN  GOTO 35
10
3520 RETURN
4000 REM **GRAFICO EM TELA CHEI
A**
4010 BORDER 0: PAPER 0: INK 7:
CLS: LET hi=0: LET lo=0
4020 FOR n=1 TO z
4030 IF a(n)>hi THEN  LET hi=a(
n)
4040 IF a(n)<lo THEN  LET lo=a(
n)
4050 NEXT n
4060 LET ra=hi-lo: LET ft=175/r
a: LET org=(ra-hi)*ft
4070 LET wd=INT (25/z)
4080 PLOT 56,org: DRAW 198,0
4090 PLOT 55,0: DRAW 0,175
4100 PRINT #1; PAPER 1;AT 0,14;
" ";x$;" "
4110 LET z$=" "+y$+" "
4120 FOR n=1 TO LEN z$
4130 PRINT AT n+(19-LEN y$)/2,0
; PAPER 1;z$(n)
4140 NEXT n
4150 PRINT AT 0,0;hi;AT 21,0;lo
4200 GOTO 3400
```

```
10 CLS:RESTORE
20 DATA ENTRAR DADOS,GRÁFICO DE
 BARRAS,VER/EDITAR DADOS,FIM DE
 PROGRAMA
30 LOCATE7,2:PRINT"M E N U   P
R I N C I P A L"
40 FOR I=1TO4:READM$
50 LOCATE10,2*I+7:PRINTI;"- ";M
$
60 NEXT
80 PRINT:PRINT:PRINTTAB(20)"OPÇ
ÃO?";
90 R$=INKEY$:IFR$=""THEN90
100 IFR$<"1"ORR$>"4"THEN90
110 ONVAL(R$)GOTO1000,2000,3000
,4000
150 LO=VA(1):HI=VA(1)
160 FOR I=1TONB%
170 IFVA(I)>HITHENHI=VA(I)
180 IFVA(I)<LOTHENLO=VA(I)
190 NEXT:RETURN
1000 CLS:IFNB%<>0THENLOCATE10,1
4:PRINT"CONFIRMA? ";ELSE1030
1010 R$=INKEY$:IFR$=""THEN1010
1020 IFR$<>"S"THEN10
1030 CLS:LOCATE10:PRINT"ENTRADA
 DE DADOS"
1040 CLEAR:LOCATE0,4:INPUT"NOME
 DO EIXO-X ";X$
1050 X$=LEFT$(X$,9):IFX$=""THEN
X$="EIXO-X"
1060 INPUT"NOME DO EIXO-Y ";Y$
1070 Y$=LEFT$(Y$,9):IFY$=""THEN
Y$="EIXO-Y"
1080 PRINT:INPUT"NUMERO DE BARR
AS: ";NB%
1090 IFNB%>65THEN1080
1100 DIMVA(NB%):PRINT:PRINT
1110 FOR I=1TONB%
1120 PRINT"VALOR DA BARRA ";I;:
INPUTVA(I)
1130 NEXT:GOTO10
2000 IFNB%=0THEN10
2010 GOSUB150
2020 IFSGN(HI)<>SGN(LO)THENHT=A
BS(HI)+ABS(LO):GOTO2050
2030 HT=ABS((SGN(HI)-1)*ABS(HI)
+(SGN(LO)=-1)*ABS(LO))
2040 IFHT=0THEN10
2050 A=2*INT((180-3*NB%)/NB%/2)
:S=3:ES=150/HT
2060 IFA>15THENA=15
2070 IFLO>=0THENY=155:GOTO2100
2080 IFHI<=0THENY=5:GOTO2100
2090 Y=INT(80+(ABS(HI)-ABS(LO))
*ES/2)
2100 SCREEN2:COLOR 15,4,4:P=61
2110 LINE (55,5)-(55,158):LINE
(52,Y)-(250,Y)
2120 FOR I=1TONB%
2130 LINE (P,Y-SGN(VA(I)))-(P+A
,INT(Y-(VA(I)*ES))),6,BF
2140 P=P+S+A:NEXT
2150 OPEN "GRP: " FOR OUTPUTAS#
1
2160 PRESET(0,3)
2170 FOR I=1TOLEN(Y$)
2180 PRINT#1,TAB(1)MID$(Y$,I,1)
:NEXT
2190 PRESET (170,160)
2200 PRINT#1,X$
2210 IFSGN(LO)<>SGN(HI)THEN2250
2220 PRESET(0,Y):PRINT#1,TAB(4)
"0"
2230 PRESET(15,155+(150*(HI>0))
):PRINT#1,LO*(-(SGN(LO)=-1))+HI
*(-(SGN(HI)=1))
2240 GOTO 2270
2250 PRESET(15,155):PRINT#1,LO:
PRESET(15,5):PRINT#1,HI
```

```
2260 PRESET(0,Y):PRINT#1,TAB(4)
"0"
2270 CLOSE
2280 IFINKEY$=""THEN2280
2290 SCREEN0:GOTO10
3000 IFNB%=0THEN10
3010 CLS:LOCATE10:PRINT"MÓDULO
DE EDIÇÃO"
3020 PRINT:PRINT"BARRA","VALOR"
3030 FORJ=1TONB%
3040 PRINTJ,VA(J)
3050 IFINT(J/17)=J/17THENGOSUB3
090
3060 NEXT
3070 GOSUB3090
3080 GOTO10
3090 LOCATE0,20:PRINT"Tecle <E>
 para editar ou qualquer outra
   tecla para continuar";
3100 R$=INKEY$:IFR$=""THEN3100
3110 IFR$<>"E"THENLOCATE0,3:FOR
H=1TO19:PRINTSPC(40):NEXT:LOCAT
E0,3:RETURN
3120 LOCATE0,20:PRINTSPC(79):LO
CATE0,20:INPUT"Qual barra ";B
3130 LOCATE0,21:INPUT"Novo valo
r ";VA(B)
3140 I=B:GOSUB150
3150 GOTO3090
4000 CLS:LOCATE7,14:INPUT"FIM D
E PROGRAMA (S/N)";R$
4010 IFR$<>"S"THEN10
```

```
10  HOME : RESTORE
20  DATA  ENTRAR DADOS,GRAFICO
DE BARRAS,VER / EDITAR DADOS, F
IM DE PROGRAMA
30  VTAB 3: HTAB 8: PRINT "M E
N U    P R I N CI P A L"
40  FOR I = 1 TO 4
50  READ M$
60  HTAB 10: VTAB 2 * I + 8: PR
INT I;"- ";M$
70  NEXT
80  PRINT : PRINT : HTAB 20: PR
INT "OPCAO ->";
90  GET R$: IF R$ < "1" OR R$ >
 "4" THEN 90
100  ON  VAL (R$) GOTO 1000,200
0,3000,4000
110 LO = VA(1):HI = VA(1)
120  FOR I = 1 TO NB%
130  IF VA(I) > HI THEN HI = VA
(I)
140  IF VA(I) < LO THEN LO = VA
(I)
150  NEXT : RETURN
1000  HOME : IF NB% <  > 0 THEN
  VTAB 15: PRINT  TAB( 10)"CONF
IRMA? (S/N)";: GET R$: IF R$ <
 > "S" THEN 10
1010  HOME : PRINT  TAB( 10)"EN
TRADA DE DADOS"
1020  CLEAR : VTAB 4: INPUT "NO
ME DO EIXO 'X': ";X$
1030 X$ =  LEFT$ (X$,5): IF X$
= "" THEN X$ = "EIXO-X"
1040  INPUT "NOME DO EIXO 'Y':
";Y$
1050 Y$ =  LEFT$ (Y$,5): IF Y$
= "" THEN Y$ = "EIXO-Y"
1060  PRINT : INPUT "NUMERO DE
BARRAS: ";NB%
1070  IF NB% > 65 THEN 1060
1080  DIM VA(NB%)
1090  PRINT : PRINT
1100  FOR I = 1 TO NB%
1110  PRINT "VALOR DA BARRA ";I
;: INPUT VA(I)
1120  NEXT : GOTO 10
2000  IF NB% = 0 THEN 10
2010  GOSUB 110
2020  IF  SGN (HI) <  > SGN (L
O) THEN HT =  ABS (HI) +  ABS (
LO): GOTO 2050
2030 HT =  ABS (( SGN (HI) = 1)
 *  ABS (HI) + ( SGN (LO) =  -
1) *  ABS (LO))
2040  IF HT = 0 THEN 10
2050 A = 2 *  INT ((270 - 2 * N
B%) / NB% / 2):S = 2:ES = 150 /
 HT
2060  IF A > 15 THEN A = 15
2070  IF LO >  = 0 THEN Y = 155
: GOTO 2100
2080  IF HI <  = 0 THEN Y = 5:
GOTO 2100
2090 Y =  INT (76 + ( ABS (HI)
-  ABS (LO)) * ES / 2)
2100  HGR : HCOLOR= 7:P = 4
2110  HPLOT 2,0 TO 2,158: HPLOT
 0,Y TO 279,Y
2120  HCOLOR= 5
2130  FOR I = 1 TO NB%
2140  FOR J = 1 TO A
2150  HPLOT J + P,Y -  SGN (VA(
I)) TO J + P, INT (Y - (VA(I) *
 ES)) + 1
2160  NEXT :P = P + S + A: NEXT

2170  VTAB 21: HTAB 1: PRINT X$
; TAB( 34);Y$
2180  VTAB 22: HTAB 10: PRINT "
VAL MAX=";HI;: HTAB 25: PRINT "
VAL MIN=";LO
2190  PRINT  TAB( 5);"PRESSIONE
 QUALQUER TECLA PARA SAIR";
2200  GET R$: TEXT : GOTO 10
3000  IF NB% = 0 THEN  GOTO 10
3010  HOME : PRINT  TAB( 10)"MO
DULO DE EDICAO"
3020  PRINT : PRINT "BARRA","VA
LOR"
3030  FOR J = 1 TO NB%
3040  PRINT J,VA(J)
3050  IF  PEEK (37) = 20 THEN
GOSUB 3090
3060  NEXT
3070  GOSUB 3090
3080  GOTO 10
3090  VTAB 21: HTAB 1: PRINT "T
ECLE <E> PARA EDITAR OU QUALQUE
R OUTRA    TECLA PARA CONTINUAR
";: GET R$: IF R$ <  > "E" THEN
  VTAB 4: HTAB 1: CALL  - 958:
RETURN
3100  VTAB 21: HTAB 1: CALL
958: INPUT "QUAL BARRA? ";B
3110  INPUT "NOVO VALOR? ";VA(B
)
3120 I = B: GOSUB 110
3130  GOTO 3090
4000  HOME : VTAB 15: HTAB 10:
PRINT "FIM DE PROGRAMA? ";: GET
 R$: IF R$ <  > "S" THEN 10
```

CRIE SPRITES NO MSX

Entre os vários recursos gráficos do MSX, o sprite destaca-se por servir de base ao funcionamento da maioria dos jogos de ação. Veja como é fácil usá-lo.

Um sprite é um tipo especial de caractere gráfico definido pelo usuário, desenhado em alta resolução e extremamente fácil de movimentar. Alguns exemplos de seu uso já foram vistos em artigos anteriores. Além da mobilidade, o sprite tem outras características especiais, tais como a possibilidade de definir e movimentar mais de um sprite e de indicar se houve colisão na tela — ou seja, se dois sprites coincidiram no mesmo ponto.

Não surpreende, portanto, que os sprites estejam presentes na maioria dos programas de jogos. Mas são, também, utilizados em qualquer tipo de programa que necessite de figuras móveis em alta resolução — por exemplo, programas financeiros que empreguem um "ícone" (símbolo apontando para as opções).

DEFINIÇÃO DE UM SPRITE

O desenho de um sprite é definido informando-se ao computador quais pontos devem ser "acesos" e quais devem ser "apagados", para que se obtenha o padrão desejado. Existem dois tamanhos de sprite: pequeno (8x8 pontos) e grande (16x16). No decorrer deste artigo, trataremos apenas dos sprites grandes, salvo menção em contrário.

Um sprite grande é formado por dezesseis linhas de dezesseis pontos cada. Em vez de se definir, ponto por ponto, quais serão acesos ou apagados, reúnem-se cada oito pontos em um grupo. Assim, a informação sobre o padrão de uma linha do sprite é armazenada em dois grupos de oito pontos cada. Atribuindo-se o algarismo "1" para os pontos acesos e "0" para os apagados, cada grupo de oito pontos se transformará em um número binário (byte de 8 bits) que, convertido para a forma decimal, dará origem a um número entre 0 e 255. Cada sprite, portanto, é formado por 32 grupos de oito pontos, 32 números binários de oito dígitos (8 bits ou 1 byte) ou, simplesmente, 32 números. Estes números são agrupados em uma linha **DATA**, ordenados como descrito adiante. Calculam-se da mesma maneira os valores para sprites pequenos, já que eles são formados por oito linhas de oito pontos cada, dando origem a 8 bytes.

A ordem dos bytes no sprite é a seguinte:

Linha 1: BYTE 1 BYTE 17
Linha 2: BYTE 2 BYTE 18
Linha 3: BYTE 3 BYTE 19

... e assim por diante até que:

Linha 15: BYTE 15 BYTE 31
Linha 16: BYTE 16 BYTE 32

... sendo que os bytes foram numerados na ordem em que aparecerão na linha **DATA**.

Uma orientação para o cálculo destes valores, uma vez definido o padrão gráfico para o sprite, é dada na figura da página 190. Por meio do exemplo, você terá uma idéia de como obter o valor decimal correspondente a um grupo de oito pontos. Se todos os pontos estiverem acesos, o valor decimal será

- O QUE É UM SPRITE?
- APLICAÇÕES
- COMO DEFINIR E MOVIMENTAR SPRITES
- PRIMEIROS PASSOS
- UM PROGRAMA PARA CRIAR SPRITES
- O USO DOS SPRITES
- "TIRO AO PÁSSARO": CONTROLE DE COLISÕES

128+64+32+16+8+4+2+1, ou seja, 255 (veja, no alto da figura, o valor pelo qual se deve multiplicar cada ponto). Se todos estiverem apagados, a soma dos valores será zero. Entre estes dois extremos haverá um único valor para cada permutação possível de pontos acesos e apagados. Cada valor obtido pode ser usado na definição do sprite, desde que colocado em uma linha **DATA**, na ordem já especificada.

OS PRIMEIROS PASSOS

Exercite-se um pouco. Comece desenhando o padrão da figura desejada em papel quadriculado. Delimite uma área de dezesseis por dezesseis quadrinhos e indique as posições dos bytes. Na figura da página 190, por exemplo, definimos um sprite que representa uma nave espacial. Ao lado da figura, temos o valor de cada byte em decimal. O primeiro byte da primeira linha tem todos os seus pontos apagados, exceto o último, cujo valor podemos encontrar no alto da figura. Como só há um ponto aceso, seu valor é o valor total do byte: 1. Na segunda linha temos o segundo byte (veja a ordem dos bytes no sprite, já fornecida). O segundo ponto está aceso, bem como o último. Se consultarmos os valores desses pontos no alta da tabela, veremos que o valor do byte 2 é 64+1=65. Os próximos três bytes têm o mesmo valor. A propósito, uma recomendação prática: após calcular o valor de um byte, verifique se, no resto do sprite, existem outros bytes com o mesmo padrão e, portanto, com o mesmo valor. Agora vamos à linha 6. O byte 6 tem o segundo e o sétimo pontos acesos, o que lhe dá um valor de 64+2=66. O valor do byte 7, que apresenta os pontos 2 e 6 acesos, é 64+4=68. O byte 8, por sua vez, tem os pontos 1, 2, 3, 5 e 8 acesos, resultando num valor de 128+64+32+8+1=233. Calcule os bytes restantes do mesmo modo. Terminado o trabalho, coloque os números obtidos em uma linha com instrução **DATA**. Para definir e movimentar o sprite, digite o programa completo:

```
10 SCREEN 2,2:COLOR 15,4,4
20 FOR I=1 TO 32
30 READ A:A$=A$+CHR$(A)
40 NEXT I
50 SPRITE$(0)=A$
60 FOR I=210 TO -20 STEP -.5
70 PUT SPRITE 1,(I,I),10,0
80 NEXT I
100 DATA 1,65,65,65,65,66,68,
233,169,191,175,163,227,3,15,17
,0,4,4,4,4,132,68,46,42,250,234
,138,142,128,224,16
```

Para a definição de um sprite em BASIC, devem-se colocar os valores já calculados dentro de uma cadeia alfanumérica. Em nosso exemplo, isso é feito pelo laço **FOR...NEXT** das linhas 20, 30 e 40. Note que dentro de uma cadeia não se colocam números e, sim, caracteres cujos códigos ASCII correspondam aos números desejados. Observe a linha 30. Ela lê os dados da linha **DATA** e os coloca na variável **A$**, usando a função **CHR$ (A)**, que define o caractere cujo código está na variável **A**.

A definição propriamente dita do sprite é feita na linha 50, pelo comando **SPRITE$ (0) A$**. O número 0 é o número do sprite (no MSX podemos ter sprites numerados de 0 a 9). Observe que na linha 10 escolhemos a tela de alta resolução, pois a de texto de 40 colunas não permite o uso de sprites. Nessa linha encontramos o comando **SCREEN 2,2**. Como você já deve saber, o primeiro número corresponde ao formato da tela (2 para a tela de alta resolução). O segundo número deve ser novo para você. Ele estabelece o tamanho do sprite utilizado: 0-pequeno (8x8), 1-pequeno ampliado (16x16, baixa resolução), 2-grande (16x16) e 3-grande ampliado (32x32, baixa resolução).

Para desenhar e movimentar o sprite na tela usamos o comando **PUT SPRITE**. Não é preciso seguir o sprite apagando posições já ocupadas: ao desenharmos um sprite na nova posição, a antiga é automaticamente apagada. O formato desse comando é **PUT SPRITE P, (X, Y), C, N**, onde **P** é a *prioridade* do sprite, **X** é a *coordenada horizontal* do sprite, variando de -32 (fora da tela, portanto) a 255; **Y** é a *coordenada vertical*, que varia de -32 a 191, podendo ainda assumir os valores 208, que faz com que todos os sprites de menor prioridade desapareçam da tela, e 209, que apaga o sprite na tela; **C** é a *cor* do sprite, e **N**, o *número* do sprite.

Quando dois ou mais sprites ocuparem a mesma posição na tela, aparecerá apenas o que estiver "na frente", ou seja, aquele que tiver o número de prioridade menor. Isso permite dar aos seus jogos efeitos tridimensionais. Os carros passam atrás dos postes, e ursos se escondem atrás de árvores, por exemplo.

		VALORES DOS PONTOS (128 64 32 16 8 4 2 1 / 128 64 32 16 8 4 2 1)	NUMEROS DA LINHA DATA	
LINHAS	1	.......X........	1	0
	2	.X.....X.....X..	65	4
	3	.X.....X.....X..	65	4
	4	.X.....X.....X..	65	4
	5	.X.....X.....X..	65	4
	6	.X....X.X....X..	66	132
	7	.X...X...X...X..	68	68
	8	XXX.X..X..X.XXX.	233	46
	9	X.X.X..X..X.X.X.	169	42
	10	X.XXXXXXXXXX.X.	191	250
	11	X.X.XXXXXXX.X.X.	175	234
	12	X.X...XXX...X.X.	163	138
	13	XXX...XXX...XXX.	227	142
	14	......XXX.......	3	128
	15	....XXXXXXX.....	15	224
	16	...X...X...X....	17	16

Na figura ao lado — um sprite representando uma nave espacial —, os números das linhas e os valores dos pontos estão indicados em suas respectivas posições, para facilitar o acompanhamento dos cálculos. Veja como os valores são somados, para dar origem aos números da linha *DATA*, mostrados à direita. Há dois números para cada linha, representando dois bytes com oito pontos cada. O cálculo desses valores pode ser feito manualmente ou com o uso de um programa.

UM PROGRAMA PARA CRIAR SPRITES

A tarefa mais aborrecida, ao se trabalhar com sprites, consiste em calcular os 32 números que serão colocados na linha **DATA**. Em vez de fazê-lo manualmente, use um programa que trabalhe por você.

Este deverá, em primeiro lugar, permitir a utilização dos recursos de edição e movimentação do cursor do MSX para desenhar a figura do sprite na tela. Em seguida, precisará calcular os valores correspondentes que serão incluídos na linha da instrução **DATA**, possibilitando-lhe usar o sprite em seus programas.

O programa seguinte foi preparado para efetuar todas essas coisas, oferecendo ainda a opção de produzir uma cópia impressa, caso você tenha uma impressora. A tarefa de digitá-lo e gravá-lo em uma fita cassete é compensadora, pois facilita muito a posterior criação de sprites.

```
5 SCREEN 0:KEY OFF
10 DIM A$(16),Z(2,16),A(16)
20 CLS:PRINT TAB(10);"EDITOR DE
 SPRITES"
30 PRINT:PRINT TAB(8);:INPUT "C
OPIA  IMPRESSA (S/N)";I$:IF I$=
"S" THEN PR$="S"
40 IF I$<>"S" AND I$<>"N" THEN
GOTO 20
50 PRINT:PRINT TAB(15);"Aguarde
"
60 FOR Z=1 TO 8
62 READ A(Z):A(Z+8)=A(Z)
64 NEXT
66 DATA 128,64,32,16,8,4,2,1
70 FOR Z=1 TO 16
75 READ A$(Z)
80 FOR ZZ=1 TO 8
85 IF MID$(A$(Z),ZZ,1)="*" THEN
Z(1,Z)=Z(1,Z)+A(ZZ)
90 NEXT ZZ:FOR ZZ=9 TO 16
95 IF MID$(A$(Z),ZZ,1)="*" THEN
 Z(2,Z)=Z(2,Z)+A(ZZ)
100 NEXT ZZ,Z
110 CLS:IF PR$="S" THEN OPEN "L
PT:" FOR OUTPUT AS #1 ELSE OPEN
 "CRT:" FOR OUTPUT AS #1
120 PRINT #1,"Linha DATA ";
130 FOR Z=1 TO 16
132 PRINT #1,Z(1,Z);",";
134 NEXT Z
136 FOR Z=1 TO 16
138 PRINT #1,Z(2,Z);:IF Z<16 TH
EN PRINT #1,",";
140 NEXT Z
145 PRINT #1," ":PRINT #1," "
150 IF PR$="S" THEN GOTO 170
155 PRINT #1,"Aperte RETURN par
a continuar"
160 K$=INKEY$
165 IF K$<>CHR$(13) THEN GOTO 1
60
170 SCREEN 1,3:COLOR 14,5,5
172 VPOKE BASE(6)+15,68
174 VPOKE BASE(6)+5,69
176 PRINT #1,TAB(5);"DESENHO DO
 SPRITE"
178 PRINT #1," ":PRINT #1," "
180 PRINT #1,TAB(5);"1234567890
123456"
190 FOR Z=1 TO 16
195 PRINT #1,TAB(5);
200 FOR ZZ=1 TO 16
205 IF MID$(A$(Z),ZZ,1)="*" THE
N PRINT #1,CHR$(120); ELSE PRIN
T #1,".";
210 NEXT ZZ:PRINT #1," ";Z
220 NEXT Z:IF PR$="S" THEN CLOS
E:END
230 FOR Z=1 TO 16
240 A$=A$+CHR$(Z(1,Z))
250 B$=B$+CHR$(Z(2,Z))
260 NEXT:SPRITE$(0)=A$+B$
270 FOR Z=190 TO 5 STEP -5
275 PUT SPRITE0,(10,Z),1
280 NEXT
290 IF PR$<>"S" THEN GOTO 290
300 REM  1234567890123456
301 DATA"                "
302 DATA"                "
303 DATA"                "
304 DATA"                "
305 DATA"                "
306 DATA"                "
307 DATA"                "
308 DATA"                "
309 DATA"                "
310 DATA"                "
311 DATA"                "
312 DATA"                "
313 DATA"                "
314 DATA"                "
315 DATA"               / "
316 DATA"                "
```

Rode o programa, tal como ele está, para verificar se houve algum erro de digitação. Ao fazê-lo, responda N à pergunta da linha 30.

Para usar o programa, digite **LIST 300-**. Isso fará com que as dezesseis linhas **DATA** vazias do final da listagem surjam na tela. Desenhe dentro delas o padrão do seu sprite. Cada uma das dezesseis posições das linhas, entre as aspas, representa um ponto. Use apenas os controles de movimentação do cursor e a barra de espaço, colocando um asterisco nas posições que deseja acender. Evite usar **INS** e **DEL**, que alteram o número de posições entre as aspas. Não deixe de apertar **RETURN** em cada uma das dezesseis linhas **DATA**; caso contrário, o MSX não se lembrará delas. Oriente-se pelos números da linha 300, que correspondem às posições horizontais. Um desenho típico se encontra na figura da página ao lado, que representa um pássaro voando.

Quando completar seu desenho, apague a tela e liste as últimas linhas para conferir. Rode o programa. O computador lhe dará, então, a opção de cópia impressa. Responda N e veja o resultado na tela.

Primeiro, surgem os 32 números das linhas **DATA**. Um desenho ampliado de

MICRO DICAS

PLANEJAMENTO DE SPRITES

Existe um modo muito prático de usar o programa de criação de sprites. Desenhe seu sprite em papel quadriculado com tamanho correspondente ao das linhas **DATA** do final da listagem. Pregue-o com fita adesiva na tela de sua TV. Em seguida, simplesmente use os controles do cursor para desenhar asteriscos nas posições dos quadrinhos que você pintou.

seu sprite é mostrado a seguir, apertando-se a tecla **RETURN**. As teclas **CTRL** e **STOP**, pressionadas ao mesmo tempo, interrompem o programa. Caso tenha uma impressora e deseje uma cópia impressa, rode o programa novamente e responda S à pergunta da linha 30. O resultado será enviado à impressora, e não à tela. Algumas impressoras requerem um ajuste no número máximo de caracteres por linha (use quarenta).

Para o exemplo da figura do pássaro, o resultado é o seguinte:

```
Linha DATA  128 , 192 , 176 , 7
2 , 116 , 84 , 50 , 42 , 41 , 3
7 , 26 , 4 , 56 , 199 , 98 , 28
 , 7 , 30 , 42 , 84 , 84 , 168
, 168 , 208 , 160 , 248 , 4 , 3
4 , 222 , 33 , 144 , 72

     DESENHO DO SPRITE


     12345678901234567
     x.............xxx   1
     xx...........xxxx.  2
     x.xx.....x.x.x.     3
     .x..x....x.x.x..    4
     .xxx.x...x.x.x..    5
     .x.x.x..x.x.x...    6
     ..xx..x.x.x.x...    7
     ..x.x.x.xx.x....    8
     ..x.x..xx.x.....    9
     ..x..x.xxxxxx...    10
     ...xx.x......x..    11
     .....x....x...x.    12
     ..xxx...xx.xxxx.    13
     xx...xxx..x....x    14
     .xx...x.x..x....    15
     ...xxx...x..x...    16
```

O mesmo programa pode ser utilizado para sprites pequenos. Para tanto, desenhe seu padrão utilizando apenas os oito primeiros bytes, ou seja, no quadrante superior esquerdo do conjunto de linhas **DATA**. Considere apenas os primeiros oito números calculados.

COMO USAR O SPRITE

O programa seguinte desenha e movimenta o sprite do pássaro, que acabamos de calcular.

```
10 SCREEN 1,2:COLOR 15,4,4
20 FOR I=1 TO 32
21 READ A:A$=A$+CHR$(A)
22 NEXT I:TIME=0
23 SPRITE$(0)=A$
25 X=90:Y=X:GOTO 50
30 K$=INKEY$:A=0:XX=0
31 IF K$="" THEN GOTO 30
32 IF K$=CHR$(30) THEN A=1:GOTO
 50
35 IF K$=CHR$(31) THEN A=2:GOTO
 50
40 IF K$=CHR$(29) THEN XX=-2:GO
TO 50
45 IF K$=CHR$(28) THEN XX=2:GOT
O 50
50 FOR Z=1 TO 10
52 X=X+XX:IF X>255 THEN X=0
55 IF X<-32 THEN X=255
60 IF A=1 AND Y>-32 THEN Y=Y-2
65 IF A=2 AND Y<190 THEN Y=Y+2
70 PUT SPRITE 0,(X,Y),15
75 NEXT Z:GOTO 30
100 DATA 128,192,176,72,116,84,
50,42,41,37,26,4,56,199,98,28,7
,30,42,84,84,168,168,208,160,24
8,4,34,222,33,144,72
```

O sprite é desenhado na tela de textos de 32 colunas e pode ser movimentado pelas teclas de controle do cursor. Para usar qualquer outro sprite, basta mudar a linha 100.

O laço contido nas linhas 20, 21 e 22 lê os valores da linha 100 e os coloca na variável **A$**. A linha 25 define o sprite atribuindo-lhe o número 0. As linhas de 30 a 75 movimentam o sprite conforme a tecla pressionada.

COLISÃO DE SPRITES

Outro recurso interessante dos sprites é a indicação da ocorrência de colisão entre dois deles, em sua movimentação pela tela. Para entender como isso é feito, apague a linha 32 do último programa e acrescente:

```
20 SPRITE ON:FOR I=1 TO 40
22 NEXT I:TIME=0:S=0
23 SPRITE$(0)=LEFT$(A$,32)
24 SPRITE$(1)=RIGHT$(A$,8)
25 X=90:Y=X:XX=100:YY=XX
30 FOR Z=1 TO 5:LOCATE 0,0
31 PRINT "TEMPO: ";TIME;
35 A=INT(RND(1)*3)+1:X=X+10
36 IF X>255 THEN X=-30
40 IF A=1 AND Y>-30 THEN Y=Y-10
45 IF A=2 AND Y<250 THEN Y=Y+10
50 PUT SPRITE 1,(X,Y),15,0
52 Z$=INKEY$:IF Z$="" THEN GOTO
 52
53 IF Z$=CHR$(29) AND XX>-30 TH
EN XX=XX-5
55 IF Z$=CHR$(28) AND XX<250 TH
EN XX=XX+5
60 IF Z$=CHR$(30) AND YY>-30 TH
EN YY=YY-5
65 IF Z$=CHR$(31) AND YY<190 TH
EN YY=YY+5
70 PUT SPRITE 0,(XX,YY),10,1
72 ON SPRITE GOSUB 1000
75 LOCATE 20,0:PRINT "SCORE: ";
S
76 NEXT Z:IF TIME<5000 THEN 30
80 PRINT:PRINT "O Tempo acabou
!":END
110 DATA 8,8,62,8,8,0,0,0
1000 SPRITE OFF:S=S+1
1010 FOR I=Y TO 191
1020 PUT SPRITE 1,(X,I),15,0
1030 NEXT I:SPRITE ON
1040 X=-30:RETURN
```

MICRO DICAS

COMO EDITAR UMA LINHA DATA

Para evitar o trabalho de digitar a linha **DATA** criada pelo programa, faça o seguinte: assim que o computador fornecer os números, pare o programa usando **CTRL** + **STOP**. Movimente o cursor até o topo da tela, onde se encontra a linha **DATA**. Usando **DEL**, apague a palavra "Linha" e, usando **INS**, escreva o número da linha **DATA** no programa que está preparando. A seguir, aperte **RETURN**. O efeito será idêntico ao da digitação de todos aqueles números.

Este método pode ser usado para colocar a mesma linha em programas já existentes em fita. Carregue o programa sem apagar a linha **DATA** da tabela. O programa carregado apaga a memória, mas não apaga a tela. Movimente o cursor até a linha **DATA**, modifique seu número, se necessário, e aperte **RETURN**.

Este é um pequeno jogo de "tiro ao pássaro", extremamente simples, mas útil à nossa finalidade. Não explicaremos seu funcionamento enquanto jogo, restringindo-nos apenas ao controle de colisões.

Para que possamos trabalhar com colisões de sprites, devemos "habilitar" o programa a reconhecer tal ocorrência, por meio da instrução **SPRITE ON**, na linha 20. Uma vez feita a habilitação, devemos usar a instrução **ON SPRITE GOSUB**, que desvia o programa para uma sub-rotina sempre que quisermos verificar se houve uma colisão de sprites. Nesta sub-rotina pode estar programado um efeito visual ou sonoro de explosão, por exemplo. Em nosso programa, tal comando ocorre na linha 72. Caso haja colisão entre o pássaro e a "mira", o controle é transferido para a sub-rotina 1000, que incrementa o escore e faz com que o pássaro caia. Durante a execução da sub-rotina, o programa é desabilitado pela instrução **SPRITE OFF** na linha 1000. Esse é um procedimento usual e quase sempre necessário, e não há uma explicação simples para ele. Retire este comando, bem como o de habilitação da linha 1030, e veja como ocorrem erros no escore.

Existe ainda a instrução **SPRITE STOP**, que não desabilita o programa mas adia qualquer desvio, guardando a ocorrência de colisões na memória. Só haverá o desvio após a ocorrência de um **SPRITE ON** no programa.

CONJUNTOS: CAIXAS DE INFORMAÇÃO

Os conjuntos constituem um recurso muito eficaz para armazenar e processar listas de qualquer natureza: nomes, datas, as mais diversas estatísticas. Aprenda a utilizá-los.

Os conjuntos são uma forma de representação de dados em programa BASIC, constituindo método mais adequado para se programar a manipulação de informações inter-relacionadas — por exemplo, listas dos membros de um clube, com seus respectivos endereços e inscrições, registros financeiros ou, então, listas de personagens, armas e tesouros para um jogo de aventuras.

O comando **LET**, que você já conhece, poderia ser utilizado para atribuir valores como esses a uma série de variáveis. No entanto, para listas muito longas, ele é bastante ineficiente, pois toma muito tempo de digitação.

O conjunto simplifica o trabalho, armazenando a informação de uma forma mais compacta. Em vez de uma variável diferente para cada item da informação, utiliza-se uma variável com o mesmo nome para todos os itens — o **A**, por exemplo. E, para se diferenciar um item dos demais, acrescenta-se simplesmente um número (ou, às vezes, uma expressão aritmética) entre parênteses, logo após o nome da variável. O primeiro elemento é chamado de **A(1)**, o segundo de **A(2)**, o terceiro de **A(3)** e assim por diante.

O conjunto não apenas torna o sistema de armazenagem mais compacto, como também facilita grandemente a programação da entrada, modificação e listagem dos dados nele contidos.

COMO DIMENSIONAR UM CONJUNTO

Antes de utilizar um conjunto em BASIC, é necessário declarar ao computador o tamanho (número de elementos) que ele o terá. Assim, o computador poderá reservar espaço suficiente na memória. Isto é feito mediante a declaração **DIM** (de "dimensão"), como se vê a seguir:

```
10 DIM A(3)
```

Use um A maiúsculo para os micros da linha ZX-81.

```
10 DIM a(4)
```

A declaração **DIM** acima informa ao computador que o conjunto **A** terá quatro elementos, ou variáveis. Nos computadores Apple, TRS-Color e MSX eles são numerados de **A(0)** a **A(3)**. Nos micros da linha Sinclair, que não aceitam o elemento **A(0)**, eles são numerados de **A(1)** a **A(4)**.

O número de elementos que se vai dimensionar pode ser tão grande quanto se queira (milhares, por exemplo), desde que caiba na memória RAM do computador, é claro. Na verdade, não é obrigatório dimensionar o número exato de elementos que se utilizarão: para ter a garantia de que o espaço será suficiente, reserve um pouco mais. Mas tenha em mente que o espaço de memória reservado com um **DIM** não poderá

- DEFINIÇÃO E UTILIZAÇÃO DE UM CONJUNTO
- COMO DIMENSIONAR UM CONJUNTO: A DECLARAÇÃO DIM
- ATRIBUIÇÃO DE VALORES
- CONJUNTOS DE NOMES
- UTILIZAÇÃO DOS DADOS EM UM CONJUNTO
- ANÁLISE DAS INFORMAÇÕES

ser ocupado por outras variáveis. Por isso, não dimensione exageradamente, ou receberá uma mensagem de "estouro de memória" (*out of memory*).

ATRIBUIÇÃO DE VALORES

O passo seguinte consiste em atribuir os valores para cada elemento. Se, por exemplo, as variáveis representassem as locações da tela nas quais se imprimiria um texto ou um gráfico, a próxima linha seria mais ou menos isto:

```
20 LET A(0)=0:LET A(1)=2:LET A(
2)=10:LET A(3)=20
```

```
20 LET a(1)=0: LET a(2)=2:
LET a(3)=10: LET a(4)=20
```

```
20 LET A(1)=0
30 LET A(2)=2
40 LET A(3)=10
50 LET A(4)=20
```

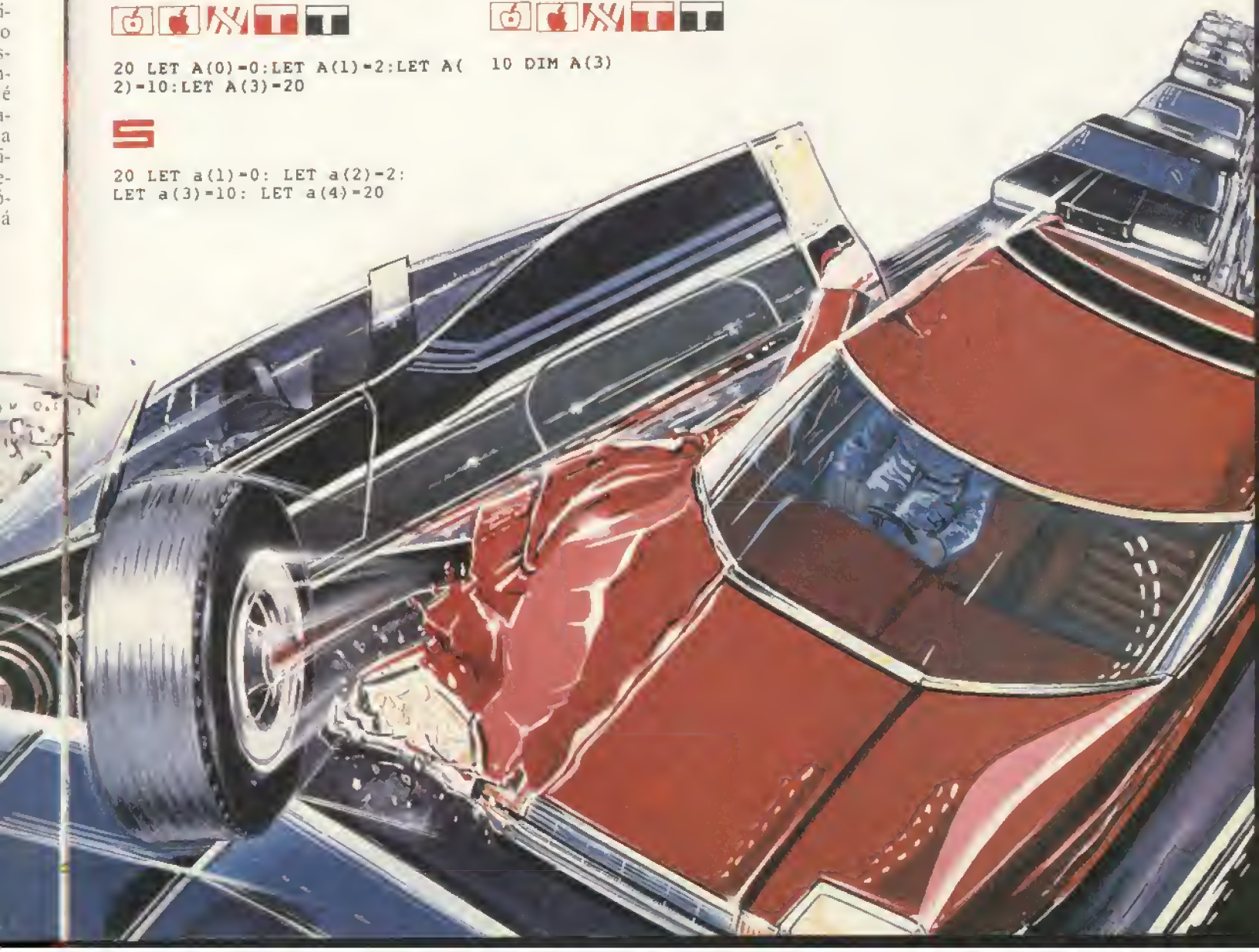
Até agora, não houve economia de tempo ou de espaço na memória. Ao contrário, teria sido bem mais rápido utilizar um simples comando **LET**. Mas veja em seguida o que acontece quando é necessário armazenar valores que são diferentes, cada vez que se roda o programa:

```
10 DIM A(3)
```

```
20 PRINT"QUAIS SAO OS VALORES?"
30 INPUT A(0),A(1),A(2),A(3)
```

```
10 DIM a(4)
20 PRINT "Quais sao os valore
s?"
30 INPUT a(1),a(2),a(3),a(4)
```

```
10 DIM A(4)
20 PRINT "QUAIS SAO
OS VALORES?"
```



```
30 INPUT A(1)
40 INPUT A(2)
50 INPUT A(3)
60 INPUT A(4)
```

O computador pedirá o valor de cada variável, sempre que você rodar o programa. Bastará, então, digitar um número (seguido por <ENTER> ou <RETURN>) quando o computador solicitar.

Assim, na medida em que o programa tiver um número maior de elementos, o tempo economizado passará a compensar o trabalho inicial. Suponha, por exemplo, que você queira entrar uma centena de números. Um programa bem simples será o suficiente:

```
10 DIM A(99)
20 FOR N=0 TO 99
30 INPUT A(N)
40 NEXT N
```

No ZX-81 digite em letras maiúsculas.

```
10 DIM a(100)
20 FOR n=1 TO 100
30 INPUT a(n)
40 NEXT n
```

CONJUNTO DE NOMES

O tipo de conjunto descrito até aqui permite apenas o armazenamento de números. Mas também é possível utilizar conjuntos para armazenar cadeias alfanuméricas.

Se você quiser processar tanto nomes quanto números, por exemplo, precisará definir dois conjuntos separadamente: um para os nomes e outro para os números. O dimensionamento do conjunto específico para nomes é assim:

```
10 DIM A$(5)
```

Nos micros da linha Sinclair (ZX-81 e Spectrum) é necessário ainda especificar o comprimento máximo que a cadeia alfanumérica poderá ter. Assim, se o maior nome da lista tiver dez caracteres, os conjuntos de nomes serão dimensionados da seguinte maneira:

No ZX-81 digite em letras maiúsculas.

```
10 DIM a$(6,10)
```

Em cada caso o computador reservou espaço na memória para seis cadeias alfanuméricas (nomes, rótulos, etc.). Para definir o laço de entrada desses nomes, digite:

```
20 FOR N=0 TO 5
30 INPUT A$(N)
40 NEXT N
```

```
20 FOR n=1 TO 6
30 INPUT a$(n)
40 NEXT n
```

Rode o programa e entre os nomes à medida que forem pedidos na tela, seguindo-os por <ENTER> ou <RETURN>. Se você quiser verificar, logo após digitar cada nome, se o armazenamento foi correto, adicione esta linha ao programa acima e rode-o novamente:

```
35 PRINT A$(N)
```

```
35 PRINT a$(n)
```

Não é difícil digitar seis nomes, mesmo que sejam longos. E, ainda que o número de nomes chegue a cem, o trabalho será rápido. Imagine, por exemplo, que você esteja realizando uma pesquisa sobre o campeonato de Fórmula 1 e deseja armazenar os nomes das curvas e o número de batidas que ocorreram em cada uma delas durante uma temporada de corridas. Não será improvável que precise entrar algumas dezenas de curvas, ou até mais, dependendo do caso. Mas o princípio é o mesmo que o utilizado no programa a seguir, com apenas seis nomes. Este programa faz com que o computador leia a série de nomes de curvas do autódromo de Jacarepaguá, armazenados em declarações **DATA** (os micros da linha ZX-81 não dispõem desse recurso):

```
10 DIM A$(5)
20 FOR N=0 TO 5
30 READ A$(N)
40 NEXT N
50 DATA NORTE,NONATO,PACE
60 DATA UM,VITORIA,LAGOA
```

```
10 DIM a$(6,11)
20 FOR n=1 TO 6
30 READ a$(n)
40 NEXT n
50 DATA"NORTE","NONATO","PACE"
60 DATA"UM","VITORIA","LAGOA"
```

Os nomes são lidos nas declarações **DATA** na linha 50 e armazenados em

um conjunto, na linha 10. Assim, cada vez que o programa é rodado, os mesmos nomes entram automaticamente. No caso de existirem cem nomes, em vez de seis, modifique a linha 10 para **DIM A$(99)** e a linha 20 para **FOR N = 0 to 99** (para os micros Apple, TRS-80, TRS-Color e MSX), ou **DIM A$ (100, 11)** e **FOR N = 1 TO 100** (para o Spectrum). Em seguida, você poderá acrescentar os 94 nomes restantes às declarações **DATA**, a partir da linha 50.

O próximo passo será a definição do conjunto que conterá o número de acidentes em cada curva.

```
60 DIM A(5)
```

```
70 FOR N=0 TO 5
80 READ A(N)
90 NEXT N
100 DATA 0,2,5,1,3,6
```

```
 60 DIM a(6)
 70 FOR n=1 TO 6
 80 READ a(n)
 90 NEXT n
100 DATA 0,2,5,1,3,6
```

É possível ignorar o elemento zero de um conjunto, nos computadores Apple, MSX, TRS-80 e TRS-Color, se for mais conveniente. Assim, no último exemplo, podemos definir **DIM A$ (6)** e numerar os itens de 1 a 6. Isso significa que **A$(0)** fica vazio, pois é mais usual contar a partir do 1 do que 0.

UTILIZAÇÃO DOS CONJUNTOS

Agora o computador já "sabe" quantos acidentes ocorreram em cada curva. Mas como essa informação pode ser manipulada?

Provavelmente, você gostaria que, antes de mais nada, o computador imprimisse uma lista com todas as curvas e o número dos acidentes registrados em cada uma — somente para checar se os dados foram digitados corretamente. Adicione as linhas seguintes ao programa e rode-o de novo.

```
210 FOR N=0 TO 5
220 PRINT A$(N),A(N)
230 NEXT N
```

```
210 FOR n=1 TO 6
220 PRINT a$(n),a(n)
230 NEXT n
```

As linhas 210 e 230 percorrem a lista, enquanto a linha 220 imprime os nomes e os números armazenados nos conjuntos correspondentes. Se algum erro de digitação foi cometido, você terá a oportunidade de localizá-lo.

Você pode, também, querer analisar os resultados contidos nos conjuntos. Por exemplo, para o levantamento das corridas, seria interessante obter respostas para questões tais como: "quantas colisões ocorreram ao todo?"; "quais são as curvas mais seguras?". As linhas adicionais do programa, destinadas a identificar o total de acidentes, são as seguintes:

```
300 CLS
310 LET TL=0
320 FOR N=0 TO 5
330 LET TL=TL+A(N)
340 NEXT N
350 PRINT "NUMERO TOTAL DE ACID
ENTES ";TL
```

Substitua no programa anterior o comando **CLS** por **HOME**, na linha 300.

```
300 CLS
310 LET total=0
320 FOR n=1 TO 6
330 LET total=total+a(n)
340 NEXT n
350 PRINT "Numero total de aci
dentes: ";total
```

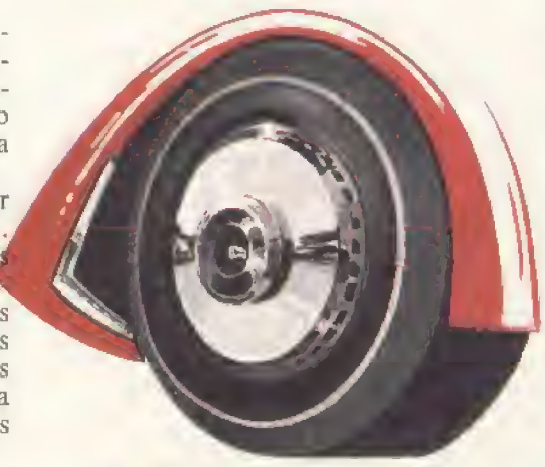
ANÁLISE DA INFORMAÇÃO

Imagine o quanto o programa acima seria útil se existissem cem curvas, em vez de apenas seis.

E, levando em conta que a informação não apenas pode ser facilmente armazenada em um conjunto, como também pode ser analisada com a mesma facilidade, você entenderá por que os conjuntos constituem um instrumento tão poderoso — para qualquer coisa, desde orçamento doméstico até finanças internacionais.

Digite estas linhas extras para examinar um exemplo de análise:

```
400 FOR N=0 TO 5
410 IF A(N)>3 THEN PRINT A$(N),
A(N)
420 NEXT N
```



```
400 FOR n=1 TO 6
410 IF a(n)>3 THEN  PRINT a$(n
 ),a(n)
420 NEXT n
```

Esta parte do programa mostra na tela uma lista das curvas nas quais ocorreram mais de três acidentes. Em nosso exemplo de seis nomes, elas são as curvas PACE e LAGOA. Se você substituir o número 3 pelo 5, na linha 410, e rodar o programa, apenas LAGOA será impresso. Para qualquer número maior do que 6 — nesse caso, o maior valor — nada será impresso.

A informação armazenada nos conjuntos poderia ser, para dar um segundo exemplo, uma lista de famílias de uma cidade, juntamente com estatísticas como o número de crianças, os rendimentos mensais e o número de carros.

Uma vez que esses elementos tenham sido entrados, eles poderão ser classificados em grupos e analisados de inúmeras maneiras. Por exemplo, suponha que queiramos listar os dados de todas as famílias com sobrenome começando com P. Para ter uma idéia de como isso funciona, acrescente estas linhas ao programa:

```
600 FORN=0TO5
620 IFLEFT$(A$(N),1)="P"THENPRI
NTA$(N)
630 NEXT
```

```
600 FOR n=1 TO 6
620 IF a$(n,1)="P" THEN  PRINT
a$(n)
630 NEXT n
```

As linhas 600 e 630 definem um laço, para examinar cada elemento do conjunto de nomes. À medida que isso ocorre, a linha 620 checa o primeiro caractere. Se ele for P, um nome inteiro aparecerá.

Nesse caso, a curva chamada PACE será impressa, porque é a única que começa com P. No exemplo do conjunto de família, poderíamos escrever uma linha do programa que listasse todas as famílias 'SILVA', todas as que tivessem mais de um carro, e assim por diante. Outra possibilidade é o cruzamento de informações — por exemplo, quais as pessoas cujos nomes começam por 'A', que moram em determinada rua, no número 21, e cujo cachorro é um fila brasileiro!

TRADUÇÃO MANUAL DO ASSEMBLY

Rápidos e eficientes, os programas em código de máquina têm, contudo, uma desvantagem: é extremamente difícil escrevê-los e depurá-los. Freqüentemente, de fato, eles parecem apenas uma seqüência de números sem sentido, pois instruções, dados e endereços são todos especificados como números hexadecimais.

A solução não está em deixar de escrever programas em códigos de máquina. A maioria deles é escrita inicialmente em linguagem Assembly e depois traduzida para código de máquina. Normalmente, isto é feito por um outro programa, chamado Assembler (ou seja, montador). Porém, se você não dispuser de um Assembler — ou se quiser aprofundar seus conhecimentos — poderá fazer essa tradução manualmente e digitar os códigos hexadecimais resultantes na memória do computador, entrando o programa monitor ou o interpretador BASIC (comandos **PEEK** e **POKE**).

A LINGUAGEM ASSEMBLY

Você terá, assim, que aprender a linguagem Assembly, que está longe de ser tão difícil quanto os códigos de máquina. Os mnemônicos, que representam os códigos de operações da máquina, praticamente dispensam explicações, pois são abreviaturas muito claras dos comandos correspondentes (mnemônico significa que lembra algo). Os dados e endereços são todos numéricos, assim como os códigos de máquina. Mas, usando códigos alfabéticos para escrever os comandos, a seqüência pura de números será rompida, e ficará muito mais fácil para o programador entender o programa resultante.

Ao se programar em linguagem Assembly, portanto, basta consultar em uma tabela os códigos mnemônicos referentes ao microprocessador utilizado pelo seu computador.

O Sinclair Spectrum, o ZX-81, o TRS-80 e o MSX empregam o microprocessador Zilog Z-80, enquanto o TRS-Color usa o Motorola 6809, e o Apple II e o TK-2000 utilizam o 6502. O resultado da consulta à tabela é o código de operação numérico correspondente. Coloque-o no lugar correto, e a tradução estará praticamente feita.

Há algo mais a ser lembrado. Nos computadores das linhas Sinclair, Apple II, TRS-80 e MSX, não se pode esquecer de trocar a ordem dos bytes constantes de um endereço de memória, do programa, ou de outros dados que precisam ser representados por dois bytes (dezesseis bits). A razão disso é que essas máquinas armazenam números no formato de byte-baixo/byte-alto. O TRS-Color utiliza um formato inverso, ou seja, de byte-alto/byte-baixo, de modo que você poderá deixar os dados ou endereços de dezesseis bytes em sua ordem normal.

OS MNEMÔNICOS

Os mnemônicos — assim como os códigos de máquina em hexa que eles representam — servem para manipular o conteúdo de um registro, definir o valor de um sinalizador ou passar de um ponto a outro no programa. Estas são, basicamente, as únicas coisas que as instruções em código de máquina são capazes de realizar. Um exemplo de mnemônico é **LDA**, que significa **LoaD Accumulator** (*carregar o acumulador*).

Se quisermos utilizar esse recurso, deveremos começar por aprender a traduzir os mnemônicos da linguagem Assembly para códigos hexadecimais.

ENDEREÇAMENTO

Contudo, traduzir programas em linguagem Assembly para código de máquina não é tão fácil quanto parece. Mesmo uma instrução simples como **LDA** pode ser traduzida para cinco a quinze códigos de operações diferentes, dependendo da máquina.

Os diversos códigos de operações dependem do tipo de endereçamento que está sendo utilizado, ou seja, das formas de acesso à memória RAM que são utilizadas pelo microprocessador.

Veremos a seguir como realizar a tradução correta de um programa em linguagem Assembly para os vários tipos de computador. Na próxima lição, aprenderemos a montar rotinas muito úteis em linguagem Assembly.

SINCLAIR

O tipo mais simples de endereçamento no ZX-81 é o chamado *endereçamen-*

Montar programas em código de máquina pode ser um risco para seu equilíbrio mental! Uma alternativa mais saudável é escrevê-los em Assembly e depois traduzi-los para hexadecimal.

- LINGUAGEM ASSEMBLY
- CÓDIGOS MNEMÔNICOS
- COMO CONVERTER OS CÓDIGOS
- ENDEREÇAMENTO
- DESVIOS E USO DE RÓTULOS

to imediato, no qual um valor numérico é usado como argumento do código de operação:

LD A,4

significa: "carregar (**LOAD**) o acumulador (**A**) com o número 4"; essa instrução é traduzida para o código **3E04** em linguagem de máquina.

Já no *endereçamento direto* é fornecido um endereço onde o dado pode ser encontrado, ao invés do próprio dado. Por exemplo:

LD A,(0E2D)

significa: "carregar o conteúdo da locação **0E2D** no registro **A**". Isto se traduz para **3A 2D 0E** (observe que os dois bytes correspondentes ao endereço foram trocados).

O endereçamento direto também funciona de modo inverso:

LD (0E2D),A

significa: "carregar o conteúdo do registro **A** na locação de memória **0E2D**". Isto é traduzido para **32 2D 0E** (novamente os dois bytes foram intercambiados). Um terceiro tipo é o *endereçamento indireto*. Este diz à máquina onde encontrar o endereço do dado necessário. Por exemplo, a instrução:

LD A, (HL)

significa: "carregar o registro **A** com o dado pertencente ao endereço contido no par de registros **HL**".

Em outras palavras, o microprocessador examina o registro **HL** e usa esse número como o endereço da locação de memória onde está o byte a ser carregado no acumulador.

Esse comando funciona também na direção contrária:

LD (HL),A

significa: "carregar o conteúdo do registro **A** na locação de memória cujo endereço se encontra no par de registros **HL**".

Existe um tipo especial de endereçamento indireto que é chamado de *endereçamento indexado*. Aqui, um dos dois registros de indexação — **IX** e **IY** — e o próprio endereço a ser utilizado são fornecidos por um valor de deslocamento, o qual é acrescentado aos conteúdos dos registros **IX** ou **IY**.

Uma instrução típica poderia ser:

LD A, (IX+2F)

Observe que o deslocamento é de apenas um byte.

Os dados também podem ser transferidos de um registro para outro. Isso é chamado de *endereçamento de registro a registro*, cuja instrução correspondente é escrita assim:

LD D,B

que significa: "carregar o conteúdo do registro **B** no registro **D**".

O *endereçamento relativo* é empregado apenas com comando de desvio. Ele diz ao computador quantos bytes este deve saltar para a frente ou para trás. Por exemplo:

JRNZ 0FC

O mnemônico **JRNZ** (*Jump Relative on Non-Zero*) significa: "saltar **FC** bytes em relação à posição atual, se o resultado da última operação efetuada for diferente de zero", ou seja, se o sinalizador de zero no registro de sinalizadores (flag register) não for igual a 1. O **FC** diz para onde saltar.

FC é −4 em complemento de dois. Assim, se o sinalizador de zero não estiver ligado, o microprocessador salta quatro bytes para trás no programa, contados a partir do final da instrução **JRNZ 0FC**, e esta é traduzida para **20 FC** em código de máquina. Assim, o microprocessador salta de fato para a instrução que apareceu dois bytes antes dela.

Na realidade, raramente o endereçamento relativo é utilizado em linguagem Assembly. Normalmente os saltos são indicados por *rótulos* (*labels*). Estas são palavras inventadas pelo programador, que cumprem a função de marcadores — tal como **INICIO** — destinados a assinalar o começo de determinadas porções de um programa.

Esse marcador aparece na frente da instrução para onde será realizado o salto, ou ainda, logo após a instrução de salto. Por exemplo, a linha inicial poderia ser:

INICIO LD A,07

LDA,(&ZD)

e em algum outro ponto do programa apareceria a instrução de salto:

DJNZ INICIO

O **DJNZ** (*Decrement and Jump on Non-Zero*) significa: "reduzir de 1 o valor do registro **B** e saltar para a instrução logo a seguir, onde o rótulo **INICIO** estiver colocado, no caso de o sinalizador de zero não estar ligado". Ao traduzir manualmente para código de máquina, você precisará elaborar os saltos relativos por sua própria conta.

A forma mais simples de endereçamento no microprocessador 6502 é o *endereçamento implícito*. Na verdade, ele não é propriamente um endereçamento. Por exemplo:

CLC

significa: "limpar o sinalizador de transporte (*Clear Carry Flag*)". Não é necessário nenhum tipo de endereço como argumento. A ação é executada no sinalizador de transporte, cujo endereço está implícito na instrução.

No endereçamento imediato o dado segue diretamente a instrução. Por exemplo:

LDA #&04

Essa operação carrega o acumulador com o número 4. O comando **LDA** é traduzido, na tabela de códigos de operações do 6502, para **A9** quando esta modalidade de endereçamento é utilizada. Assim, a instrução completa passa a ser **A904**.

Já no *endereçamento absoluto*, o endereço completo de uma locação de memória segue o mnemônico. Por exemplo:

LDA & 1122

significa: "carregar o acumulador com o dado armazenado na locação de memória **1122**". Essa forma também é conhecida como endereçamento direto.

A tradução para o **LDA** com endereçamento absoluto é o código hexadecimal **AD**. No exemplo acima, a instrução completa é traduzida para **AD 2211**. Observe que essa operação tem três bytes e que os dois bytes do endereço são trocados de ordem, quando se traduz da linguagem Assembly para o Apple.

Na página 0 — isto é, de 0000 a 00FF — não é preciso especificar o primeiro byte de endereço, mas deve-se utilizar um código de operação especial de página 0 (chamado de código de operação de endereço curto), o qual diz ao computador para olhar para um endereço de um byte.

O código de operação para **LDA** na modalidade de endereçamento de página 0 é A5. Assim, uma instrução como:

LDA &7F

será traduzida para **A57F**.

Com o endereço absoluto e o endereço de página 0, é possível utilizar o *endereçamento indexado*, no qual o conteúdo de um dos registros de indexação **—X** e **Y—** é acrescentado ao endereço dado com a instrução, para fornecer um segundo endereço que será utilizado. Por exemplo:

LDA &1122,X

Suponha que o conteúdo de registro **X** seja 33. Acrescentando-se 33 a 1122, obtém-se 1155; o acumulador é, então, carregado com o dado que se encontra na locação de memória 1155.

Ambos os registros **X** e **Y** podem ser empregados para indexar tanto o endereçamento absoluto quanto o de página 0.

Mas note que, se a soma do conteúdo do registro **X** com o endereço da página 0 for maior do que FF e, em virtude disso, cair na página 1, o byte mais significativo será ignorado. No endereçamento de página zero, indexado ou qualquer outro, o endereço utilizado sempre estará na página 0.

Ao usar endereçamento indexado deve-se consultar o código mnemônico em Assembly, correspondente à página 0X, página 0Y, absoluto X ou absoluto Y.

Também é possível endereçar indiretamente uma locação de memória, utilizando-se o endereçamento indireto. Nesta forma, o código de operação é seguido por um endereço entre parênteses. Isso significa que o microprocessador encontrará nesta locação um segundo endereço, o qual será empregado para acessar o dado. Por exemplo:

JMP (&1530)

significa: salte para o endereço dado na locação de memória 1530. Mas como qualquer locação de memória pode reter apenas um byte, e como são necessários dois bytes para compor um endereço, o microprocessador olha para 1530 e 1531. A primeira locação contém o byte menos significativo, e a segunda, o byte mais significativo, de acordo com a convenção utilizada para esse microprocessador. Assim, se a locação de memória 1530 contém 2F, e a locação de memória 1531 contém 13, o microprocessador saltará para a locação de memória 132F.

Existem também duas maneiras de indexar endereços indiretos. Com o registro **X** você poderá acrescentar um deslocamento ao primeiro endereço (o endereço fornecido na instrução). O procedimento é chamado de *endereçamento indireto pré-indexado*. Alternativamente, você poderá acrescentar um deslocamento do registro **Y** para o segundo endereço (o endereço nas locações de memória fornecido na instrução original). Isso é chamado de *endereçamento indireto pós-indexado*.

Esta é a aparência de instruções deste tipo, em linguagem Assembly:

LDA (&1122,X) e LDA (&1122),Y

O primeiro código usa o endereçamento pré-indexado, e o segundo, o pós-indexado. Para se consultar os códigos hexadecimais correspondentes, deve-se procurar o **LDA** sob **(indireto,X)** e **(in-**

3A2D0E

direto),Y. Isso fornecerá **A1** e **B1** respectivamente. Assim, as duas instruções acima seriam traduzidas para **A1 2211** e **B1 22 11**.

As instruções de desvio são saltos condicionais. Por exemplo:

BEQ

significa: desviar se for igual a (**B**ranch if **EQ**ual) — isto é, se o sinalizador zero estiver igual a 1. As instruções de desvio podem utilizar endereçamento relativo em alguns tipos de Assembler.

BEQ #&04

significa: salte quatro bytes para a frente a partir do início da próxima instrução, se o sinalizador zero estiver igual a 1.

Por meio de rótulos, indica-se ao microprocessador o ponto do programa para onde o desvio deve ser realizado. A primeira instrução a partir deste ponto é simplesmente precedida pelo rótulo. Por exemplo:

INICIO LDA &04

ao passo que a instrução que produz o desvio terá:

BEQ INICIO

O Assembler se encarregará, então, de calcular o desvio relativo. No entanto, se você está traduzindo manualmente uma listagem em Assembly, a responsabilidade pelo cálculo será sua. Código de máquina não emprega rótulos, apenas números. Os microprocessadores 6510 e 6502 comportam ainda mais um tipo de endereçamento — o *endereçamento por acumulador* —, usado principalmente com instruções de deslocamento e rotação de bits. Por exemplo:

ASL A

significa: desloque o acumulador de um bit (**A**ccumulator **S**hift **L**eft). É possível utilizar esta instrução, também, com outras locações de memória, e não apenas com o acumulador. Neste caso, basta substituir o A, no exemplo acima, pelo endereço de memória cujo conteúdo você quer deslocar. Esta locação pode ser indexada com o registro **X**, apenas.

A instrução tem o seguinte efeito: o bit menos significante da memória é zerado, o bit mais significante é colocado no registro indicador de transporte (*carry flag*), e os bits restantes são deslocados uma posição à esquerda. Da mesma forma, existe o **ASR** (**A**ccumulator **S**hift **R**ight), para efetuar deslocamentos de bits à direita.

T

O microprocessador Motorola 6809, utilizado nos computadores compatíveis com a linha TRS-Color, tem um tipo de endereçamento implícito, também chamado *endereçamento de registros*.

Esta expressão refere-se às instruções da linguagem Assembler que não precisam ser acompanhadas de um endereço pois operam em um registro interno do microprocessador, que é estipulado pela própria instrução. Por exemplo:

DECA

significa: diminua de 1 o registro **A** (**D**ecrement **A**ccumulator). O código hexadecimal correspondente, que pode ser encontrado na tabela de conversão do manual do microprocessador, é 4A.

No endereçamento imediato, o dado a ser utilizado é o próprio argumento da instrução. Por exemplo:

ADDB #$7

significa: adicione 7 ao registro **B**. O código para ADD em modo imediato é CB, em hexadecimal, de modo que a instrução completa acima é traduzida para CB 07.

Existe ainda outra forma de endereçamento imediato, na qual se usa um segundo byte, chamado *pós-byte*. Neste caso, a instrução assume a forma:

TFR A,B

que significa: transferir o conteúdo do registro **A** para o registro **B**.

Para traduzir essa instrução para código de máquina, deve-se consultar o código TFR na seção de endereçamento imediato da tabela. Isto nos dá o código IF. A seguir, o pós-byte é avaliado um *nibble* (um grupo de quatro bits) de cada vez.

Em geral, para usar esta instrução com o 6809, atribui-se um dígito hexadecimal para cada registro. Dois desses dígitos são unidos para formar o pós-byte. Por exemplo, o registro A recebe o valor de 8, e o registro B, o valor de 9. De modo que:

TFR A,B

é traduzido para IF 89. Se, ao contrário, a instrução tivesse sido:

TFR B,A

a instrução correspondente em código de máquina seria IF 98.

No endereçamento absoluto, ou *ampliado*, a instrução é acompanhada do endereço de dezesseis bits da locação de memória a ser utilizada. Por exemplo:

STA $7530

significa: armazene o conteúdo do acumulador da memória &H7530 (**S**tore **A**ccumulator). A tradução, no caso, seria B7 75 30 (note que os dois últimos bytes, correspondentes ao endereço, são mantidos na ordem direta, como é obrigatório para a maior parte dos outros tipos de microcomputador).

A instrução é um pouco complicada, pois envolve três bytes. Usando-se endereçamento direto, porém, podemos reduzi-la para uma instrução de dois bytes. O microprocessador 6809 tem um registro de página direta, que armazena o byte mais significativo de um endereço. Assim, todas as memórias naquela página podem ser endereçadas usando-se apenas um byte, no caso o menos significante.

Esse recurso funciona da seguinte

maneira: o registro de página direta é carregado usando-se a instrução **TFR**, mencionada acima, ou a instrução **EXG** (EXchanGe), que troca os conteúdos de dois registros entre si.

EXG A,DP

Na instrução acima, pede-se para intercambiar os registros A e página direta (Direct Page). A instrução não apenas define a página direta com o que foi carregado em A com um comando **LDA** prévio; ela também armazena o número de página direta previamente retido por **DP**, de volta para o registro **A**. Daí esse número pode ser jogado para uma locação de memória qualquer, usando a instrução **STA**.

O byte mais significante da página direta — 75, neste exemplo — está agora definido, e o byte menos significante pode ser dado com a instrução. Neste caso, para fazer a tradução manual, é necessário consultar a instrução **STA** na tabela de códigos, sob a seção de endereçamento indireto. O resultado é 97, em hexadecimal. Assim, a instrução completa do exemplo será 9730.

Obviamente, não vale a pena repetir todo este procedimento cada vez que se quiser armazenar dados em uma locação de memória. Definir o endereço-base da página direta toma muito mais tempo do que o economizado com a redução da instrução de três bytes para dois bytes. Mas muitas vezes é suficiente definir a página direta no começo de um segmento de programa e, daí em diante, armazenar todos os dados na mesma página.

Endereçamento indireto ocorre quando o microprocessador precisa examinar uma memória indicada por um endereço que, por sua vez, contém um segundo endereço, no qual estão os dados que serão carregados no acumulador. A forma de notação utilizada é entre colchetes:

LDA [$FEEE]

Por meio desta instrução, pede-se ao microcomputador para recuperar um endereço, nas locações de memória FEEE e FFFF (o endereço completo, com dois bytes).

Prosseguindo, a instrução carrega no acumulador A o conteúdo na locação com este segundo endereço.

O endereço indireto também pode ser realizado com os registros **U**, **S**, **X** e **Y**, bem como com o contador de programa.

Em todos esses casos de endereçamento indireto, é necessário consultar o mnemônico da instrução na tabela, sob a seção de endereçamento indexado, para se obter o código correspondente. Em seguida, deve-se consultar o pós-byte.

No exemplo dado acima, consulta-se **LDA** sob a seção de endereçamento indexado, e se obtém A6. Depois, consulta-se o endereço [mmmm], que indica um endereço geral de dezesseis bytes, na tabela de pós-bytes. Isto dá 9 F (ou BF, ou DF ou FF, pois os pós-bytes são repetidos para cada registro: como [mmmm] é independente de qualquer registro, aparece quatro vezes).

Assim a instrução completa:

LDA [$FFFE]

é traduzida para A6 9F FF FE.

Usando os registros **U**, **S**, **X** e **Y**, bem como o contador de programa, pode-se estipular tanto o endereçamento não-indireto (sem colchetes), quanto o endereçamento direto (com colchetes), usando-se *deslocamentos* (*offsets*). Os deslocamentos podem ser constantes — decimais, hexadecimais de oito ou dezesseis bits — ou o conteúdo de outros registros. Este conteúdo é somado ou subtraído do conteúdo-base de um dos cinco endereços indexáveis.

LDA 0 ,X

significa: carregue o registro acumulador com dados na locação de memória, cujo endereço está armazenado no registro **X**. Nesse exemplo, o deslocamento é zero. Outros exemplos:

LDA 1,X

significa: carregar o acumulador com o endereço em **X**, somado de 1.

LDA −16,Y

significa: carregar o acumulador com o endereço em **Y**, diminuído de 16.

LDA ($10,X)

é a versão para endereçamento indexado. Ele carrega o acumulador com o conteúdo da locação de memória cujo endereço está indicado em **X**, e que é adicionado de 10 (em hexadecimal).

Em todos estes casos, o código operacional para **LDA** pode ser encontrado na seção de endereçamento indexado, e o pós-byte apropriado precisa ser adicionado, a partir do que for encontrado na tabela de pós-bytes. Deslocamentos de oito bits e endereços de dezesseis bits seguem-se a isto.

Desvios em linguagem Assembler são programados por meio das instruções **JMP** e **JSR**. Estes desvios podem ser definidos em termos de endereços finais de página direta, ampliados, ou, ainda, indexados, de modo que os códigos de máquina equivalentes se alteram de acordo. Desvios condicionais, entretanto, utilizam endereçamento relativo.

Com endereçamento relativo, informa-se à UCP quantos bytes o programa deve pular para a frente ou para trás em relação à instrução de desvio.

Existem dois tipos de desvios: um desvio ordinário de oito bits, que pula para até 127 bytes para a frente ou 128 bytes para trás, e um desvio longo de dezesseis bits, que pode abranger 32 767 bytes para trás.

BEQ $FA

pulará seis bytes para trás, a partir do início da instrução seguinte, se o indicador de zero (*zero flag*) estiver ligado. FA é −6 em complemento de dois. Alternativamente, temos o desvio longo:

LBEQ $0A00

que saltará 2560 bytes para a frente, se o indicador de zero estiver ligado.

Os desvios sempre utilizam endereçamento relativo — não há outro tipo de endereçamento disponível para eles. Porém, como muitas vezes é trabalhoso calcular o tamanho de um salto, procura-se utilizar rótulos.

Os rótulos são palavras (simbólicas) criadas pelo programador, que indicam ao microprocessador qual é o ponto do programa para onde o desvio deve ser realizado e, a partir deste ponto, a primeira instrução é simplesmente precedida pelo rótulo. Por exemplo:

INICIO DECA

ao passo que a instrução que produz o desvio terá:

BEQ INICIO

O Assembler se encarregará, então, de calcular o desvio relativo, que poderá ser para a frente, se o rótulo estiver depois da instrução de desvio, ou para trás, se estiver antes dela. Entretanto, se você está traduzindo manualmente uma listagem em Assembly, a responsabilidade pelo cálculo será sua, já que código de máquina não emprega rótulos, apenas números. Para fazer o cálculo, conta-se o número de bytes a partir do final da instrução de desvio até o começo da instrução para a qual se deseja saltar. Lembre-se que, para um salto longo, isto tomará dois bytes.

LINHA	FABRICANTE	MODELO
Apple II +	Appletronica	Thor 2010
Apple II +	CCE	MC-4000 Exato
Apple II +	CPA	Absolutus
Apple II +	CPA	Polaris
Apple II +	Digitus	DGT-AP
Apple II +	Dismac	D-8100
Apple II +	ENIAC	ENIAC II
Apple II +	Franklin	Franklin
Apple II +	Houston	Houston AP
Apple II +	Magnex	DM II
Apple II +	Maxitronica	MX-2001
Apple II +	Maxitronica	MX-48
Apple II +	Maxitronica	MX-64
Apple II +	Maxitronica	Maxitronic I
Apple II +	Microcraft	Craft II Plus
Apple II +	Milmar	Apple II Plus
Apple II +	Milmar	Apple Master
Apple II +	Milmar	Apple Senior
Apple II +	Omega	MC-400
Apple II +	Polymax	Maxxi
Apple II +	Polymax	Poly Plus
Apple II +	Spectrum	Microengenho I
Apple II +	Spectrum	Spectrum ed
Apple II +	Suporte	Venus II
Apple II +	Sycomig	SIC I
Apple II +	Unitron	AP II
Apple II +	Victor do Brasil	Elppa II Plus
Apple II +	Victor do Brasil	Elppa Jr.
Apple IIe	Microcraft	Craft IIe
Apple IIe	Microdigital	TK-3000 IIe
Apple IIe	Spectrum	Microengenho II
MSX	Gradiente	Expert GPC-1
MSX	Sharp	Hotbit HB-8000
Sinclair Spectrum	Microdigital	TK-90X
Sinclair Spectrum	Timex	Timex 2000
Sinclair ZX-81	Apply	Apply 300
Sinclair ZX-81	Engebras	AS-1000
Sinclair ZX-81	Filcres	NEZ-8000
Sinclair ZX-81	Microdigital	TK-82C
Sinclair ZX-81	Microdigital	TK-83
Sinclair ZX-81	Microdigital	TK-85
Sinclair ZX-81	Prologica	CP-200
Sinclair ZX-81	Ritas	Ringo R-470
Sinclair ZX-81	Timex	Timex 1000
Sinclair ZX-81	Timex	Timex 1500
TRS-80 Mod. I	Dismac	D-8000
TRS-80 Mod. I	Dismac	D-8001/2
TRS-80 Mod. I	LNW	LNW-80
TRS-80 Mod. I	Video Genie	Video Genie I
TRS-80 Mod.III	Digitus	DGT-100
TRS-80 Mod.III	Digitus	DGT-1000
TRS-80 Mod.III	Kemitron	Naja 800
TRS-80 Mod.III	Prologica	CP-300
TRS-80 Mod.III	Prologica	CP-500
TRS-80 Mod.III	Sysdata	Sysdata III
TRS-80 Mod.III	Sysdata	Sysdata Jr.
TRS-80 Mod.IV	Multix	MX-Compacto
TRS-80 Mod.IV	Sysdata	Sysdata IV
TRS-Color	Codimex	CS-6508
TRS-Color	Dynacom	MX-1600
TRS-Color	LZ	Color 64
TRS-Color	Microdigital	TKS-800
TRS-Color	Prologica	CP-400

FABRICANTE	MODELO	PAÍS	LINHA
Appletronica	Thor 2010	Brasil	Apple II +
Apply	Apply 300	Brasil	Sinclair ZX-81
CCE	MC-4000 Exato	Brasil	Apple II +
CPA	Absolutus	Brasil	Apple II +
CPA	Polaris	Brasil	Apple II +
Codimex	CS-6508	Brasil	TRS-Color
Digitus	DGT-100	Brasil	TRS-80 Mod.III
Digitus	DGT-1000	Brasil	TRS-80 Mod.III
Digitus	DGT-AP	Brasil	Apple II +
Dismac	D-8000	Brasil	TRS-80 Mod. I
Dismac	D-8001/2	Brasil	TRS-80 Mod. I
Dismac	D-8100	Brasil	Apple II +
Dynacom	MX-1600	Brasil	TRS-Color
ENIAC	ENIAC II	Brasil	Apple II +
Engebras	AS-1000	Brasil	Sinclair ZX-81
Filcres	NEZ-8000	Brasil	Sinclair ZX-81
Franklin	Franklin	USA	Apple II +
Gradiente	Expert GPC1	Brasil	MSX
Houston	Houston AP	Brasil	Apple II +
Kemitron	Naja 800	Brasil	TRS-80 Mod.III
LNW	LNW-80	USA	TRS-80 Mod. I
LZ	Color 64	Brasil	TRS-Color
Magnex	DM II	Brasil	Apple II +
Maxitronica	MX-2001	Brasil	Apple II +
Maxitronica	MX-48	Brasil	Apple II +
Maxitronica	MX-64	Brasil	Apple II +
Maxitronica	Maxitronic I	Brasil	Apple II +
Microcraft	Craft II Plus	Brasil	Apple II +
Microcraft	Craft IIe	Brasil	Apple IIe
Microdigital	TK-3000 IIe	Brasil	Apple IIe
Microdigital	TK-82C	Brasil	Sinclair ZX-81
Microdigital	TK-83	Brasil	Sinclair ZX-81
Microdigital	TK-85	Brasil	Sinclair ZX-81
Microdigital	TK-90X	Brasil	Sinclair Spectrum
Microdigital	TKS-800	Brasil	TRS-Color
Milmar	Apple II Plus	Brasil	Apple II +
Milmar	Apple Master	Brasil	Apple II +
Milmar	Apple Senior	Brasil	Apple II +
Multix	MX-Compacto	Brasil	TRS-80 Mod.IV
Omega	MC-400	Brasil	Apple II +
Polymax	Maxxi	Brasil	Apple II +
Polymax	Poly Plus	Brasil	Apple II +
Prologica	CP-200	Brasil	Sinclair ZX-81
Prologica	CP-300	Brasil	TRS-80 Mod.III
Prologica	CP-400	Brasil	TRS-Color
Prologica	CP-500	Brasil	TRS-80 Mod.III
Ritas	Ringo R-470	Brasil	Sinclair ZX-81
Sharp	Hotbit HB-8000	Brasil	MSX
Spectrum	Microengenho I	Brasil	Apple II +
Spectrum	Microengenho II	Brasil	Apple IIe
Spectrum	Spectrum ed	Brasil	Apple II +
Suporte	Venus II	Brasil	Apple II +
Sycomig	SIC I	Brasil	Apple II +
Sysdata	Sysdata III	Brasil	TRS-80 Mod.III
Sysdata	Sysdata IV	Brasil	TRS-80 Mod.IV
Sysdata	Sysdata Jr.	Brasil	TRS-80 Mod.III
Timex	Timex 1000	USA	Sinclair ZX-81
Timex	Timex 1500	USA	Sinclair ZX-81
Timex	Timex 2000	USA	Sinclair Spectrum
Unitron	AP II	Brasil	Apple II +
Victor do Brasil	Elppa II Plus	Brasil	Apple II +
Victor do Brasil	Elppa Jr.	Brasil	Apple II +
Video Genie	Video Genie I	USA	TRS-80 Mod. I

INPUT foi especialmente projetado para microcomputadores compatíveis com as sete principais linhas existentes no mercado.
Os blocos de textos e listagens de programas aplicados apenas a determinadas linhas de micros podem ser identificados por meio dos seguintes símbolos:


Sinclair ZX-81


TRS-80


TK-2000


MSX


Spectrum


TRS-Color


Apple II

Quando o emblema for seguido de uma faixa, então tanto o texto como os programas que se seguem passam a ser específicos para a linha indicada.

NO PRÓXIMO NÚMERO

PROGRAMAÇÃO DE JOGOS

Crie suas próprias aventuras e descubra inúmeras dicas que poderão torná-las mais envolventes e excitantes.

CÓDIGO DE MÁQUINA

Se você deseja maior intimidade com o computador, exercite-se na tradução do Assembly para código de máquina.

PROGRAMAÇÃO BASIC

Aprenda a utilizar conjuntos bidimensionais para uma eficiente armazenagem de dados inter-relacionados.